НИКОЛАЙ ПРОХОРЕНОК

bhv®

Разработка Web-сайтов с помощью Perl и MySQL

- ОБРАБОТКА ДАННЫХ ФОРМЫ
- ПОЛУЧЕНИЕ ИНФОРМАЦИИ ИЗ ИНТЕРНЕТА
- ОПИСАНИЕ ВСТРОЕННЫХ ФУНКЦИЙ MySQL
- ИСПОЛЬЗОВАНИЕ ПРОГРАММЫ phpMyAdmin
- ПУБЛИКАЦИЯ САЙТА В ИНТЕРНЕТЕ
- ПРИМЕРЫ И СОВЕТЫ ИЗ ПРАКТИКИ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Николай Прохоренок

Разработка Web-сайтов с помощью Perl и MySQL

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.06
ББК 32.973.26-018.2
П84

Прохоренок Н. А.

П84 Разработка Web-сайтов с помощью Perl и MySQL. — СПб.: БХВ-Петербург, 2009. — 560 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0377-8

На практических примерах описана разработка динамических Web-сайтов с помощью Perl и MySQL. Рассмотрены основные конструкции языка Perl, даны приемы написания сценариев, наиболее часто используемых при разработке Web-сайтов. Уделено внимание способам работы с базами данных посредством Perl, а также вопросам администрирования баз с помощью программы phpMyAdmin. Показано, как обрабатывать данные формы, отправлять письма с сайта, загружать файлы на сервер с помощью формы, создавать личный кабинет для пользователей, гостевую книгу, форум и др.

Для Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 04.12.08.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 45,15.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0377-8

Оглавление

Введение

Добро пожаловать в мир Perl!

Perl (Practical Extraction and Report Language, практический язык извлечения данных и отчетов) — это интерпретируемый язык программирования, созданный Ларри Уоллом (Larry Wall) в 1987 г. для обработки текстовой информации и написания различных системных программ в операционной системе UNIX. Легкость изучения языка и бесплатное распространение сделали Perl очень популярным языком программирования. На сегодняшний день, Perl обязательно входит в стандартную поставку операционной системы UNIX и реализован для большинства других операционных систем (например, Windows, Macintosh). Чаще всего Perl применяется для администрирования UNIX и Web-программирования, хотя возможности этого языка гораздо шире.

Основная цель этой книги — познакомить читателя с Web-программированием и технологией CGI (Common Gateway Interface). Так как Perl предоставляет широчайшие возможности по обработке текстовой информации, то он является идеальным инструментом для создания Web-приложений. В настоящее время Perl считается основным средством написания CGI-программ и доступен практически на всех серверах, работающих под управлением операционной системы семейства UNIX (например, FreeBSD). А это более 90 % всех серверов в Интернете!

Эта книга научит создавать динамические Web-сайты с помощью Perl и MySQL, без использования специализированных редакторов. Ведь только при работе с текстом (например, в Блокноте Windows) можно досконально прочувствовать весь процесс и стать настоящим программистом. Если вы уже строили статические сайты с помощью HTML и CSS и хотите научиться своими руками создавать динамические Web-сайты, свободно владеть Perl и MySQL, то эта книга для вас.

Что же можно создать с использованием изучаемых технологий? Давайте рассмотрим возможности этих технологий, а также назначение глав книги.

В *главе 1* мы установим на компьютер специальное программное обеспечение, с помощью которого можно будет создавать и тестировать программы, написанные на языке Perl. Это позволит изучить основные конструкции языка и удалить все ошибки из скриптов до их загрузки на сервер. Ведь сайт может стать очень популярным, и посетителям не очень понравится увидеть вместо необходимой информации сообщение об ошибке.

В *главе 2* мы изучим базовые понятия языка Perl. Научимся управлять программой с помощью условных операторов и циклов, создавать собственные подпрограммы и целые модули, рассмотрим принципы реализации объектно-ориентированного программирования, технологию обработки ошибок.

Изучив материал *главы 3*, вы сможете работать с файлами и каталогами, обрабатывать данные формы на сервере, сделать рассылку писем, создать личный кабинет для пользователей, загрузить файлы на сервер с помощью формы, создать гостевую книгу, форум, чат, интернет-магазин и многое другое. В этой же главе мы рассмотрим технологию доступа к базам данных MySQL и Access с помощью Perl.

В наше время ни один крупный портал не обходится без использования баз данных. С помощью MySQL можно эффективно использовать базы данных, добавлять, изменять и удалять данные, получать данные по запросу. В *главе 4* мы рассмотрим создание реляционных баз данных, изучим язык SQL и все технологии поиска информации, а также большинство встроенных функций MySQL.

В *главе 5* мы на примере рассмотрим создание динамического сайта с помощью Perl и MySQL, Личного кабинета пользователей и защиты его с помощью Perl, а также создадим Личный кабинет администратора и защитим его средствами сервера Apache. Создаваемые программы научат правильно обрабатывать данные формы и работать с базами данных.

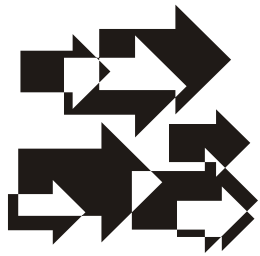
В *главе 6* мы рассмотрим основные проблемы, связанные с размещением сайта в Интернете. Научимся работать с протоколом FTP, изучим структуру каталогов сервера, произведем настройку Web-сервера Apache с помощью файла .htaccess, рассмотрим назначение файлов robots.txt и favicon.ico.

Практически все примеры являются законченными программами, которые выводят какой-либо результат. Настоятельно рекомендую обязательно рассматривать все примеры из книги и набирать код вручную. Вы сделаете при

этом множество ошибок. А умение находить эти ошибки воспитает вас настоящим программистом.

Ваши замечания и пожелания вы можете оставить на форуме сайта **http://wwwadmin.ru/**. Все замеченные опечатки прошу присылать на E-mail **mail@bhv.ru**.

Желаю приятного прочтения и надеюсь, что эта книга станет верным спутником в вашей повседневной жизни.



ГЛАВА 1

Установка программного обеспечения под Windows

1.1. Знакомьтесь — Денвер

Для тестирования и настройки программ необходимо установить на компьютер специальное программное обеспечение:

- Web-сервер Apache — программное обеспечение, отвечающее за отображение документов, запрашиваемых при наборе URL-адреса в адресной строке Web-браузера;
- интерпретатор PHP — для выполнения программ, написанных на языке PHP (например, программы phpMyAdmin);
- интерпретатор Perl — для выполнения программ, написанных на языке Perl;
- MySQL — сервер баз данных;
- phpMyAdmin — для управления базами данных.

Все программы можно бесплатно получить с сайтов производителей.

Вместо установки и ручной настройки этих программ можно установить на компьютер систему Денвер, разработанную Дмитрием Котеровым. Установка Денвера предельно проста и полностью автоматизирована. Базовый пакет Денвера включает:

- Web-сервер Apache с поддержкой SSL, SSI, mod_rewrite, mod_php;
- интерпретатор PHP5 с поддержкой GD, MySQL, sqLite;
- MySQL5;
- phpMyAdmin;

- miniPerl;
- эмулятор программы Sendmail и SMTP-сервера.

Дополнительные модули, компоненты и программы доступны в виде пакетов расширений.

Прежде чем устанавливать Денвер, необходимо проверить сетевые настройки и отсутствие программ, занимающих порты 80 и 443, т. к. эти порты использует Web-сервер Apache. Для проверки выбираем пункт меню **Пуск | Выполнить...**. В окне **Запуск программы** в поле **Открыть** набираем `cmd`, а затем нажимаем кнопку **ОК**. В командной строке набираем команду:

```
ping 127.0.0.1
```

Результат выполнения команды изображен на рис. 1.1.

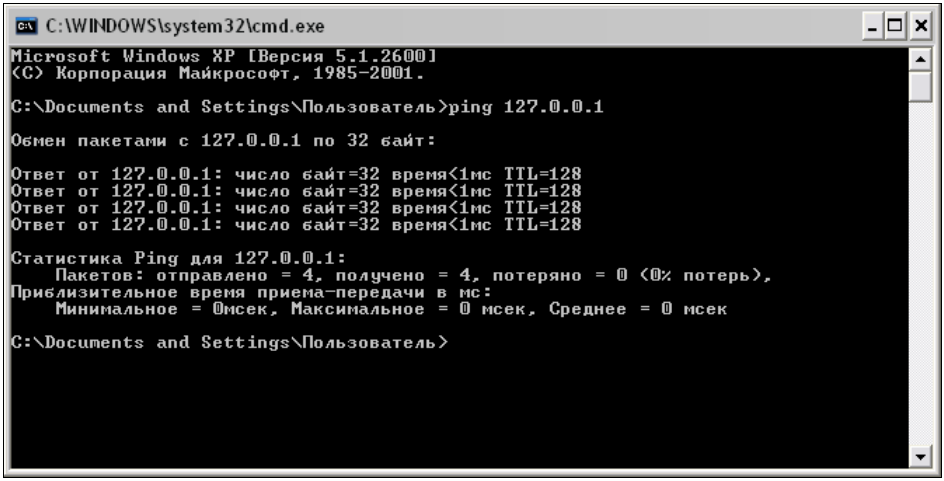


Рис. 1.1. Результат выполнения команды `ping`

Если число потерянных пакетов больше 0, то необходимо проверить сетевые настройки.

Чтобы проверить порты 80 и 443, в командной строке набираем команду:

```
netstat -anb
```

В списке не должно быть строк с портами 80 и 443. Если они есть, то Web-сервер Apache не сможет запуститься. Обычно эти порты занимают программы Skype и Web-сервер IIS. Перед установкой и использованием Денвера эти программы не следует запускать.

Если проверка прошла успешно, то можно смело устанавливать Денвер.

1.1.1. Установка Денвера

С сайта **http://www.denwer.ru/** скачиваем дистрибутив Денвера. Размер базового пакета составляет 5.5 Мбайт. Копируем на свой компьютер файл Denwer3_Base_2008-01-13_a2.2.4_p5.2.4_m5.0.45_pma2.6.1.exe или более новую версию и запускаем файл. В итоге отобразится такое окно, как на рис. 1.2.

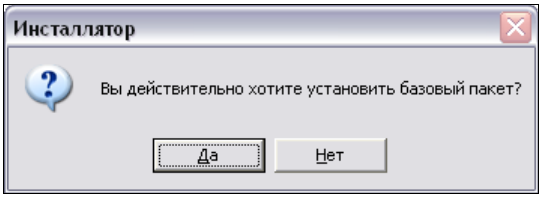


Рис. 1.2. Запуск инсталлятора

Для продолжения установки нажимаем кнопку **Да**. Откроются сразу два окна — черное окошко и окно Web-браузера (рис. 1.3).

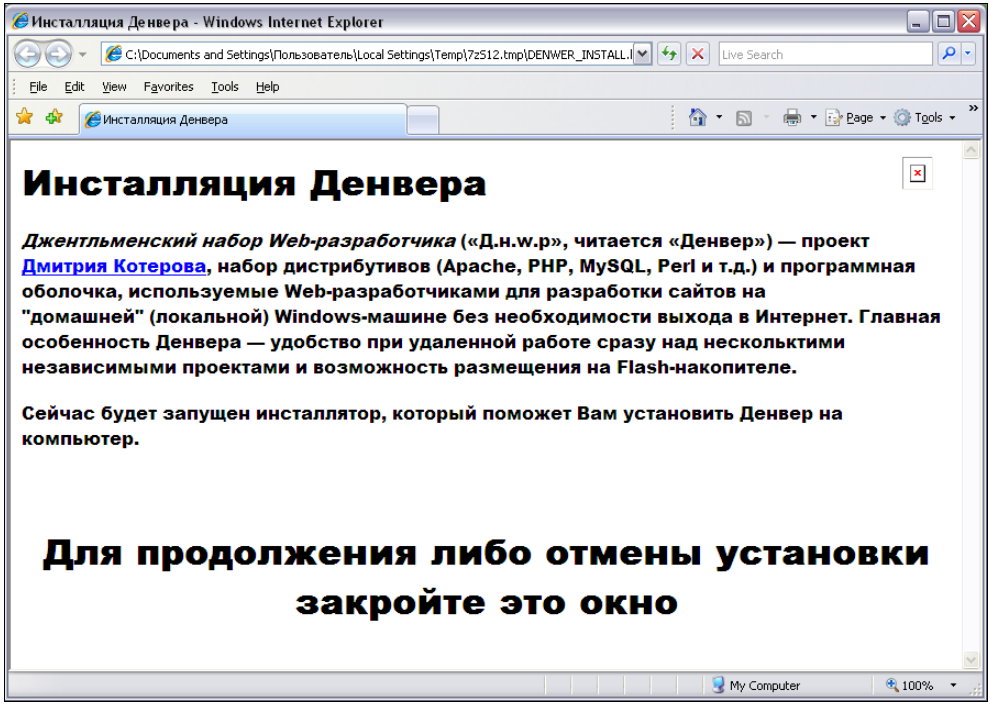


Рис. 1.3. Окно **Инсталляция Денвера**

Для продолжения установки закрываем окно Web-браузера. Если необходимо прервать установку Денвера, то нажимаем комбинацию клавиш <Ctrl>+ +<Break>. Нажимаем клавишу <Enter> для начала установки. Инсталлятор проверит наличие необходимых драйверов и утилит. Если все прошло без проблем, то будет предложено установить Денвер в папку C:\WebServers (рис. 1.4).

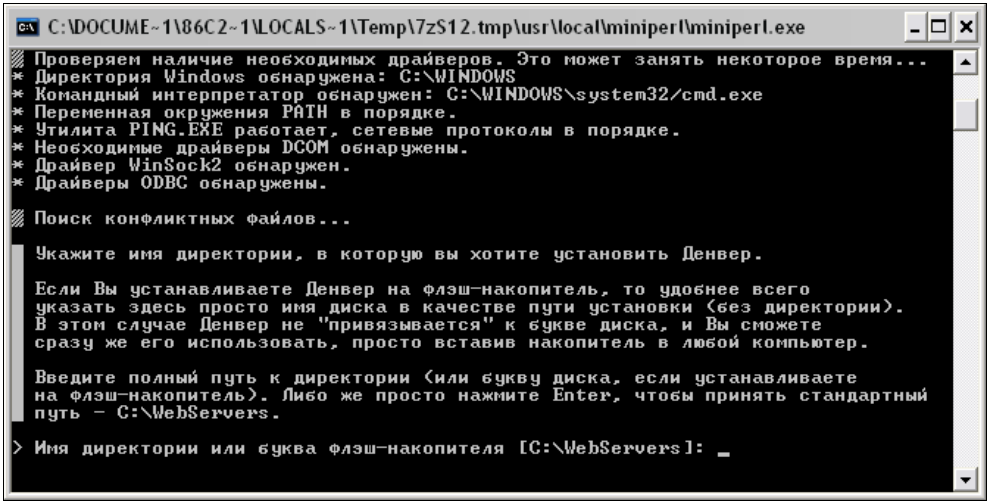


Рис. 1.4. Выбор папки для установки Денвера

На этом шаге можно изменить папку для установки. В любом случае необходимо устанавливать Денвер в каталог первого уровня, т. к. инсталлятор пакетов расширений ищет базовый пакет именно в каталогах первого уровня по всем дискам. Если базовый пакет не будет найден, то его местонахождение необходимо будет указывать вручную. Мы согласимся с каталогом по умолчанию. Нажимаем клавишу <Enter>, а затем подтверждаем установку в каталог C:\WebServers. Для этого нажимаем клавишу <y>.

При запуске Денвер создает виртуальный диск, который просто указывает на определенный каталог. Это позволяет создать на компьютере разработчика структуру каталогов, которая используется в операционной системе UNIX. Для продолжения установки нажимаем клавишу <Enter>. На этом шаге инсталлятор проверит наличие утилиты subst, необходимой для создания виртуального диска (рис. 1.5).

На следующем шаге необходимо выбрать имя будущего виртуального диска (рис. 1.6).

Обратите внимание

Диск с таким именем не должен присутствовать в системе.

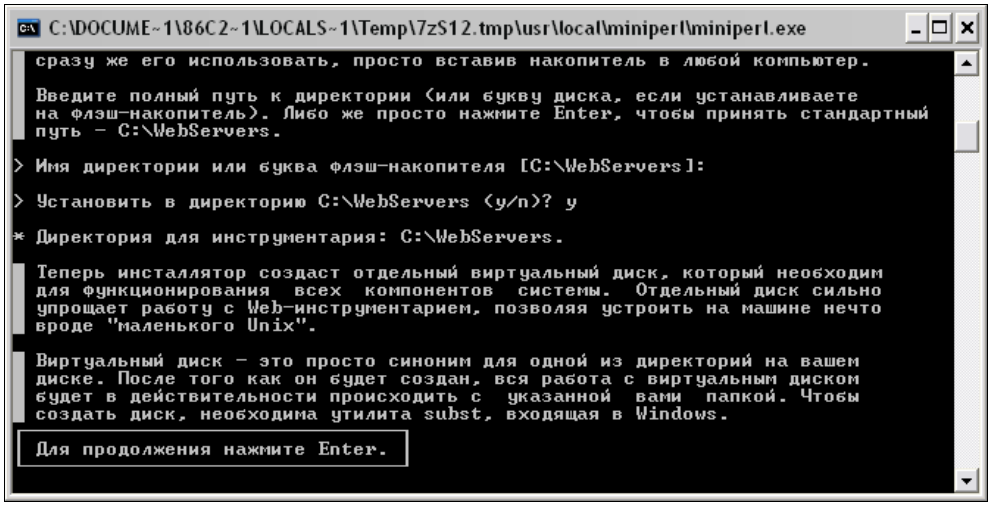


Рис. 1.5. Проверка утилиты subst

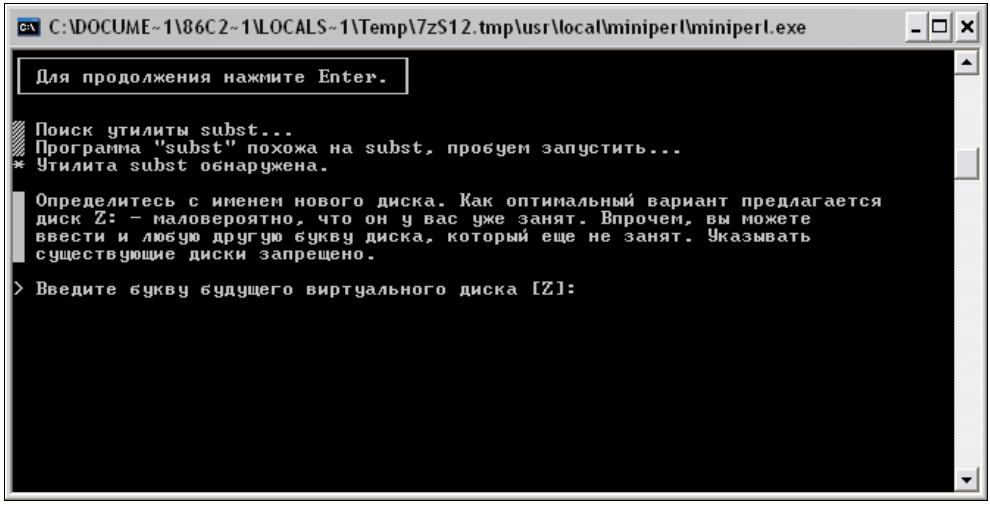


Рис. 1.6. Выбор буквы виртуального диска

По умолчанию предлагается использовать букву Z. Мы согласимся с этим именем. Для этого нажимаем клавишу <Z> или просто <Enter> (диск Z подразумевается по умолчанию). На этом шаге инсталлятор попробует создать, а затем отключить виртуальный диск (рис. 1.7).

Для начала копирования файлов в каталог C:\WebServers нажимаем клавишу <Enter>.

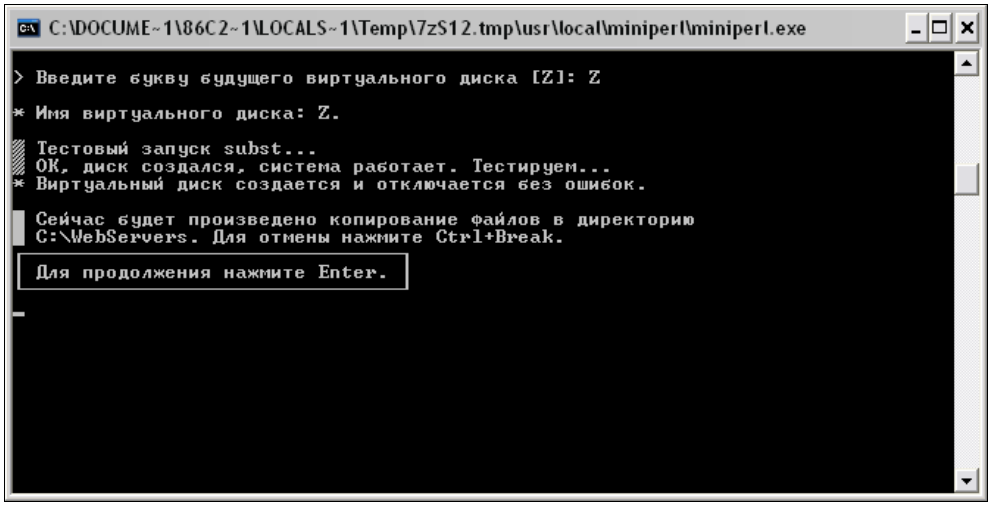


Рис. 1.7. Тестирование виртуального диска

На следующем шаге (рис. 1.8) предлагается выбрать вариант запуска Денвера. Первый вариант предполагает запуск Денвера при загрузке операционной системы. При втором варианте Денвер будет запускаться и останавливаться с помощью ярлыков на Рабочем столе. Хотя и рекомендуется выбрать первый вариант, мы выберем именно второй. В этом случае Денвер будет запускаться только тогда, когда нам это нужно. Нажимаем клавишу <2>.

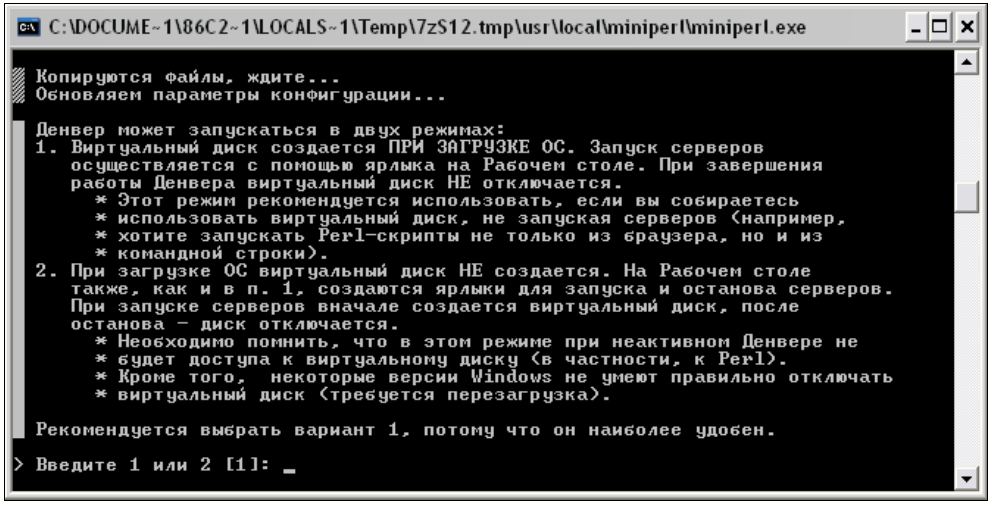


Рис. 1.8. Выбор варианта запуска Денвера

Подтверждаем желание разместить ярлыки Денвера на Рабочем столе. Для этого нажимаем клавишу <y>. В итоге отобразится окно Web-браузера с сообщением об успешной установке Денвера (рис. 1.9).

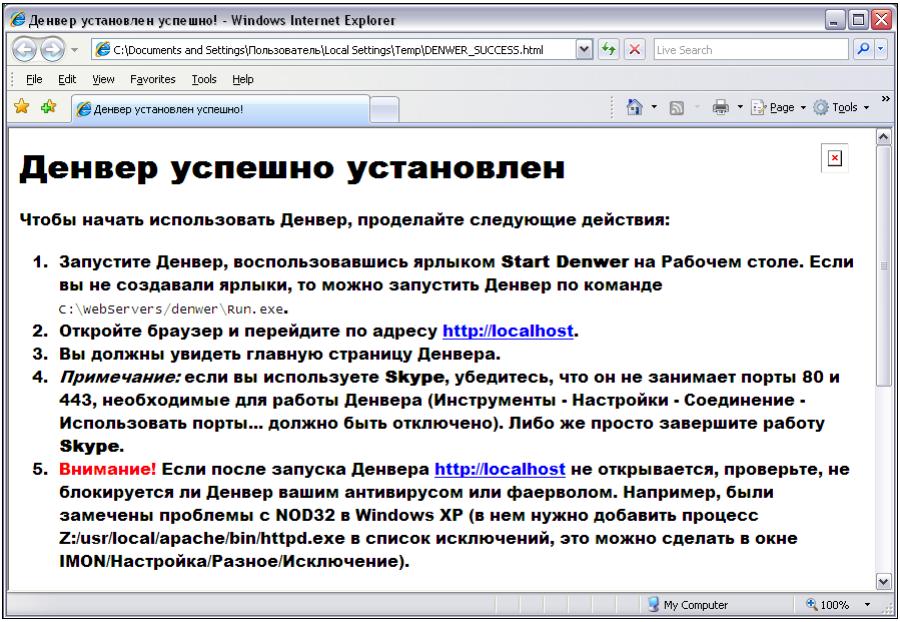


Рис. 1.9. Сообщение при успешной установке Денвера

На Рабочем столе будут созданы три ярлыка (рис. 1.10):

- **Start Denwer** — для запуска Денвера;
- **Restart Denwer** — для перезапуска Денвера;
- **Stop Denwer** — для остановки Денвера.

Рис. 1.10. Ярлыки Денвера

Запускаем Денвер с помощью ярлыка **Start Denwer** на Рабочем столе. Если на компьютере установлен брандмауэр, то при первом запуске отобразится окно **Оповещение системы безопасности Windows** (рис. 1.11).

Следует обязательно выбрать кнопку **Разблокировать**. В случае успешного запуска Денвера в правом нижнем углу отобразятся два логотипа (рис. 1.12).

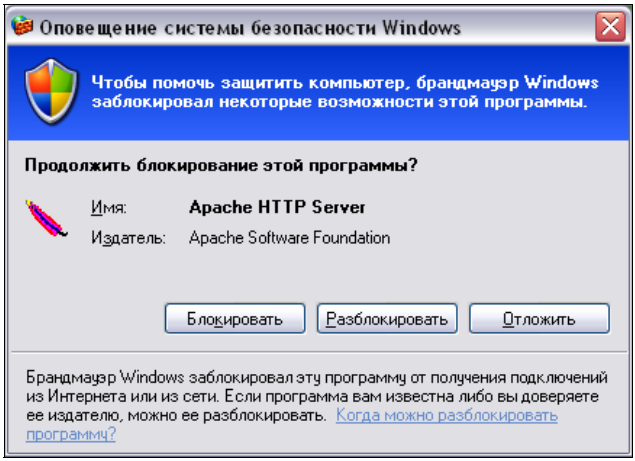


Рис. 1.11. Окно **Оповещение системы безопасности Windows**

Рис. 1.12. Логотипы Денвера и сервера Apache

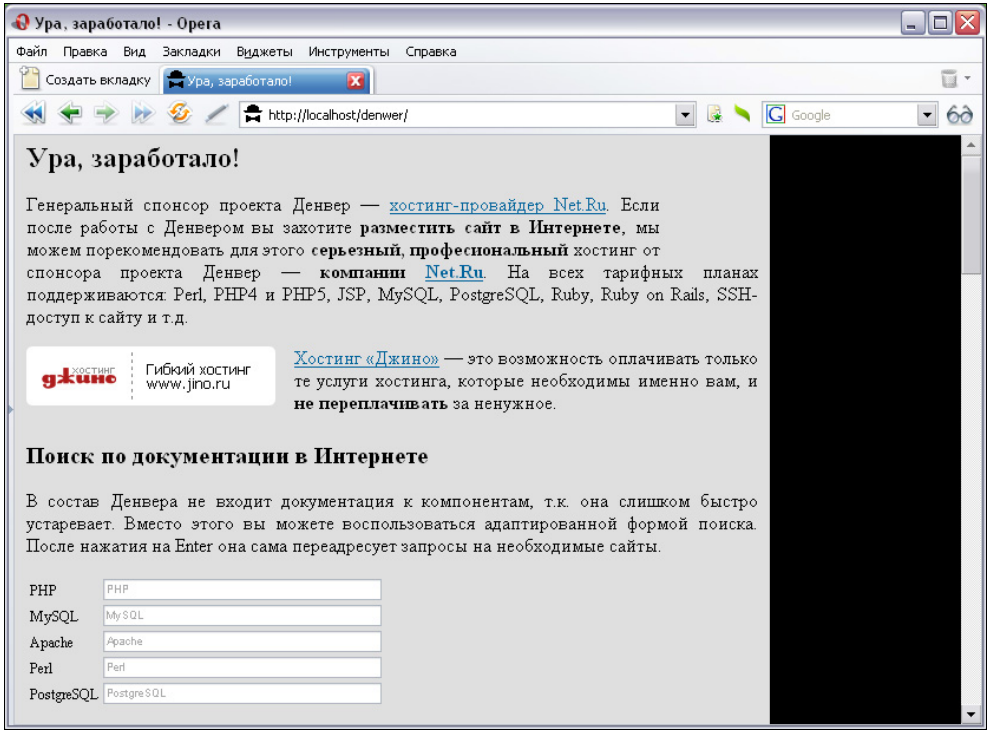


Рис. 1.13. Сообщение при успешном запуске Денвера

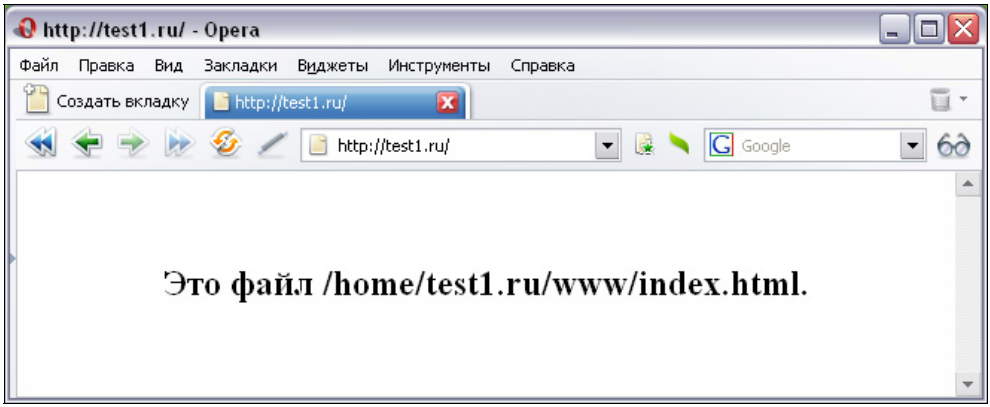


Рис. 1.14. Сообщение о тестировании хоста **http://test1.ru/**

Для проверки работоспособности Денвера в адресной строке Web-браузера набираем **http://localhost/**. Если все нормально, то отобразится окно с надписью "Ура, заработало!" (рис. 1.13).

Для проверки работоспособности виртуальных хостов в адресной строке Web-браузера задаем адрес **http://test1.ru/**. Если все нормально, то отобразится окно с надписью "Это файл /home/test1.ru/www/index.html" (рис. 1.14).

Если сообщение не появилось, необходимо проверить, запущена ли служба DNS-клиент. Это можно сделать, открыв окно **Пуск | Настройка | Панель управления | Администрирование | Службы**. В параметре **Тип запуска** напротив службы DNS-клиент должно быть значение **Авто**, а в параметре **Состояние** — значение **Работает**.

1.1.2. Установка пакетов расширений

Базовый пакет Денвера уже содержит интерпретатор Perl, однако он не содержит модулей Perl, а также имеет довольно старую версию. Так как мы планируем изучать именно Perl, то необходимо установить пакет расширения ActivePerl версии 5.8. Вместе с пакетом устанавливается инсталлятор модулей PPM, который позволит установить дополнительные модули.

Со страницы **http://www.denwer.ru/packages/perl.html** копируем пакет расширения ActivePerl. Размер дистрибутива 10 Мбайт.

> ***Обратите внимание***
>
> Прежде чем устанавливать пакет расширения, необходимо обязательно установить базовый пакет Денвера.

Запускаем файл Denwer3_Perl_2008-01-13_5.8.8.exe. В итоге отобразится окно, как на рис. 1.15.

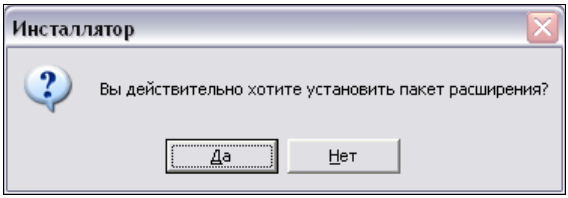


Рис. 1.15. Запуск инсталлятора

Для начала установки нажимаем кнопку **Да**. Инсталлятор произведет поиск базового пакета Денвера. Если базовый пакет найден, то будет предложено установить пакет расширения в найденный каталог (рис. 1.16).

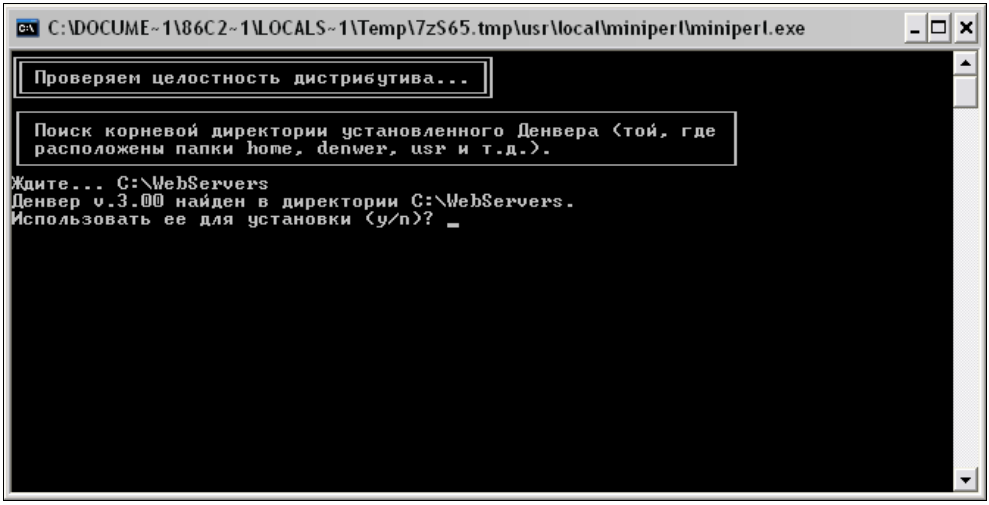


Рис. 1.16. Выбор каталога для установки пакета расширения

Подтверждаем установку пакета расширения в каталог C:\WebServers, для этого нажимаем <y>. Нажимаем <Enter> для начала установки (рис. 1.17).

В результате файлы будут скопированы в каталог C:\WebServers. Для завершения установки нажимаем клавишу <Enter>.

Далее необходимо добавить каталог с установленным интерпретатором Perl в переменную PATH операционной системы. Для этого в меню **Пуск** выбираем пункт **Панель управления** (или **Настройка | Панель управления**). В открывшемся окне выбираем пункт **Система**. Переходим на вкладку **Дополнительно** (рис. 1.18). Нажимаем кнопку **Переменные среды**. Откроется окно **Переменные среды** (рис. 1.19). В разделе **Системные переменные** делаем двойной щелчок на строке **Path** (или выделяем строку и нажимаем кнопку **Изменить**). Откроется окно **Изменение системной переменной** (рис. 1.20).

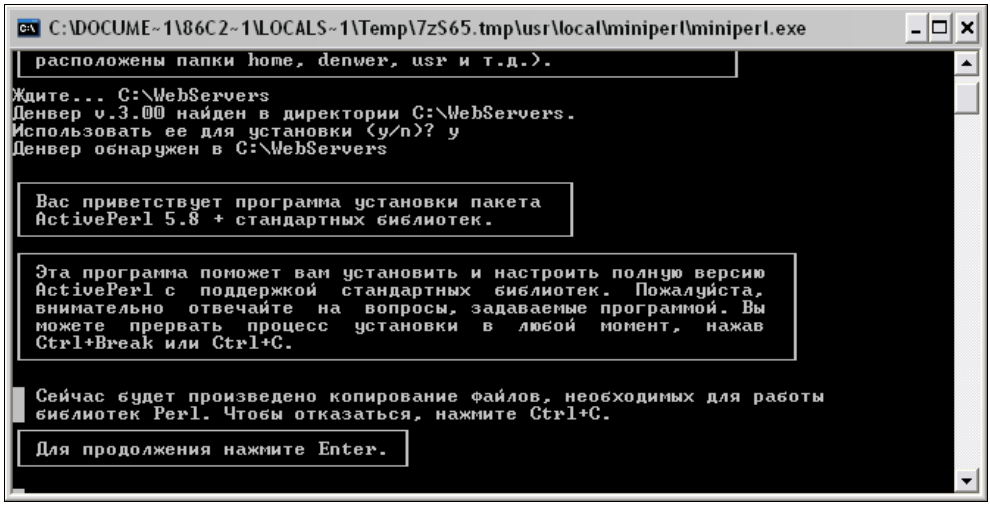


Рис. 1.17. Начало установки пакета расширения

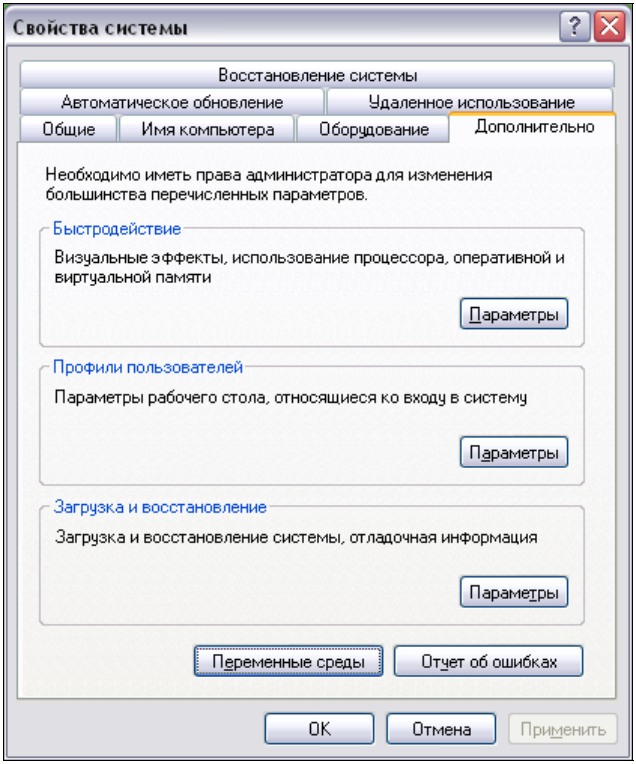


Рис. 1.18. Окно **Свойства системы**, вкладка **Дополнительно**

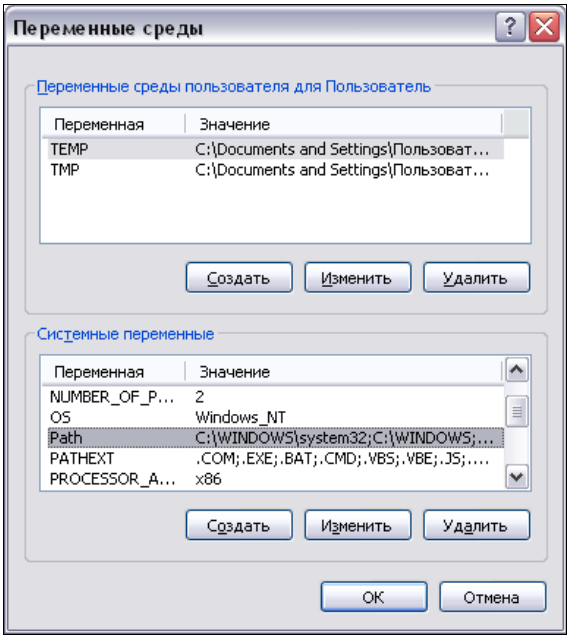


Рис. 1.19. Окно **Переменные среды**

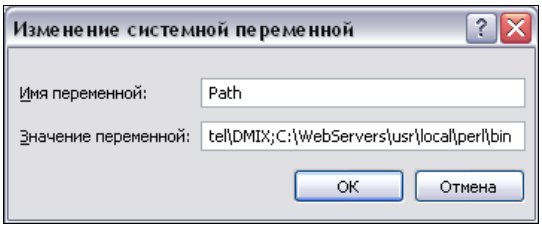


Рис. 1.20. Окно **Изменение системной переменной**

В имеющемся значении переменной `PATH` добавляем путь к каталогу с установленным Perl (C:\WebServers\usr\local\perl\bin) через точку с запятой:

```
;C:\WebServers\usr\local\perl\bin
```

Точку с запятой необходимо обязательно поставить, этот символ разделяет пути. Трижды нажимаем кнопку **OK**. После данных изменений следует перезагрузить компьютер.

1.1.3. Установка модулей

Для работы с MySQL из Perl необходимо установить модуль DBD-mysql. Для этого запускаем Денвер. Переходим на диск Z и запускаем файл Z:\usr\local\perl\bin\ppm-shell.bat.

В командной строке должно быть приглашение:

```
ppm>
```

Набираем команду:

```
install http://theoryx5.uwinnipeg.ca/ppms/DBD-mysql.ppd
```

и нажимаем клавишу <Enter>.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Файл ppm-shell.bat должен быть обязательно запущен с диска Z. В противном случае модуль будет установлен в каталог C:\usr, а не в C:\WebServers\usr.
>
> При установке модулей компьютер должен быть подключен к Интернету.

1.1.4. Запуск и остановка Денвера

Для запуска Денвера предназначен ярлык **Start Denwer** на Рабочем столе. Если по каким-либо причинам ярлык не создан, то запустить Денвер можно с помощью файла C:\WebServers\denwer\Run.exe. После запуска Денвера:

- создается виртуальный диск Z;
- запускается сервер Apache и MySQL;
- в переменную `PATH` прописывается путь к необходимым папкам;
- в файл hosts (C:\WINDOWS\system32\drivers\etc) прописываются виртуальные хосты.

Для перезапуска Денвера предназначен ярлык **Restart Denwer**. Если по каким-либо причинам ярлык не создан, то перезапустить Денвер можно с помощью файла C:\WebServers\denwer\Restart.exe. Перезапустить Денвер необходимо, например, в случае создания виртуальных хостов.

Для остановки Денвера предназначен ярлык **Stop Denwer** на Рабочем столе. Если по каким-либо причинам ярлык не создан, то остановить Денвер можно с помощью файла C:\WebServers\denwer\Stop.exe. После остановки Денвера:

- отключается виртуальный диск Z (если вы не выбрали вариант с постоянно запущенным виртуальным диском при установке Денвера);
- останавливаются Apache и MySQL;
- переменная `PATH` получает свое первоначальное значение;
- из файла hosts (C:\WINDOWS\system32\drivers\etc) удаляются виртуальные хосты, созданные Денвером.

Иными словами, после остановки Денвер не оставляет после себя никаких следов.

1.1.5. Создание виртуальных хостов

По умолчанию после установки Денвера сконфигурированы три виртуальных хоста:

- **http://localhost** — содержит скрипты тестирования и различные утилиты;
- **http://test1.ru**;
- **http://custom-host:8648** — хост, имеющий свой собственный IP-адрес и порт.

Для тестирования скриптов из книги мы создадим виртуальный хост perlbook.ru. Для этого создаем папку perlbook.ru в каталоге C:\WebServers \home. Внутри новой папки создаем каталоги:

- www — для файлов в формате HTML, PHP и картинок;
- cgi-bin — для скриптов, написанных на языке Perl.

Внутри папки www создаем файл index.html со следующим кодом:

```
<HTML>
<HEAD>
<TITLE>Новый хост</TITLE>
</HEAD>
<BODY>
Это наш новый хост.
</BODY>
</HTML>
```

Запускаем Денвер (или перезапускаем, если Денвер был запущен раньше). Открываем Web-браузер и в адресной строке набираем **http://perlbook.ru/**. В итоге должна отобразиться надпись "Это наш новый хост". Как видите, создать виртуальный хост в Денвере очень просто.

Если необходимо создать хост третьего уровня, например new.perlbook.ru, то в папке C:\WebServers\home\perlbook.ru создаем соответствующую папку. В нашем случае — с названием new.

Если необходимо создать хост четвертого уровня, например host.new.perlbook.ru, то в папке C:\WebServers\home\perlbook.ru создаем папку с названием host.new.

Список всех зарегистрированных виртуальных хостов можно увидеть, если в Web-браузере набрать адрес **http://localhost/denwer/Tools/sitelist/index.php** (рис. 1.21).

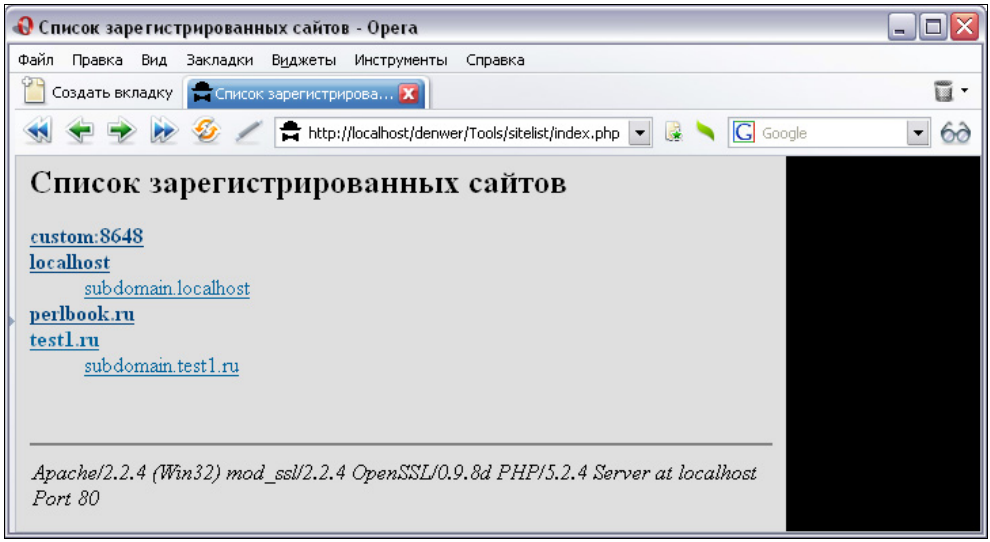


Рис. 1.21. Список всех зарегистрированных виртуальных хостов

1.1.6. Конфигурационные файлы Денвера

Рассмотрим основные конфигурационные файлы:

- CONFIGURATION.txt — это основной файл конфигурации Денвера. Расположен он в папке C:\WebServers\denwer. С помощью директивы `subst_drive` можно изменить имя виртуального диска, а с помощью директивы `runlevel` изменить тип запуска Денвера. Если указать значение `main`, то виртуальный диск будет создаваться при загрузке операционной системы. В нашем случае директива должна иметь значение `reserve`;
- httpd.conf — основной файл конфигурации сервера Apache. Расположен в папке C:\WebServers\usr\local\apache\conf;
- php.ini — основной файл конфигурации PHP. Расположен в папке C:\WebServers\usr\local\php5;
- my.cnf — основной файл конфигурации MySQL. Расположен в папке C:\WebServers\usr\local\mysql5;
- config.inc.php — файл конфигурации phpMyAdmin. Расположен в папке C:\WebServers\home\localhost\www\Tools\phpmyadmin.

1.2. Установка ActivePerl

Если у вас уже установлены Apache, PHP, MySQL и phpMyAdmin, то вместо установки Денвера следует установить только интерпретатор Perl под

Windows под названием ActivePerl. Если вы хотите установить все компоненты по отдельности, то можно воспользоваться описанием установки и настройки программ из книги "HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера" издательства "БХВ-Петербург".

Скачать программу ActivePerl можно с сайта **http://www.activestate.com/**. Из списка выбираем ActivePerl-5.8.8.822-MSWin32-x86-280952.msi или более позднюю версию. Скачиваем файл и запускаем. В итоге отобразится окно, изображенное на рис. 1.22.

Рис. 1.22. Установка ActivePerl. Шаг 1

Для начала установки нажимаем кнопку **Next**. На втором шаге необходимо принять лицензионное соглашение (рис. 1.23).

Для принятия лицензионного соглашения устанавливаем флажок **I accept** и нажимаем кнопку **Next**.

На следующем шаге можно выбрать устанавливаемые компоненты и каталог для установки (рис. 1.24).

Проверьте, чтобы был указан каталог для установки C:\Perl\. Нажимаем кнопку **Next** для продолжения установки.

В открывшемся окне (рис. 1.25) предлагается автоматически добавить путь к интерпретатору Perl в переменную `PATH` и связать расширение pl с запуском интерпретатора Perl. Первый флажок советую оставить установленным, а вот второй необходимо снять, в противном случае программы будут запускаться двойным щелчком мыши. Намного удобнее связать расширение pl с открыти-

ем текста программы в Блокноте. Иначе для редактирования программы придется из контекстного меню выбирать пункт **Открыть с помощью**. Нажимаем кнопку **Next**.

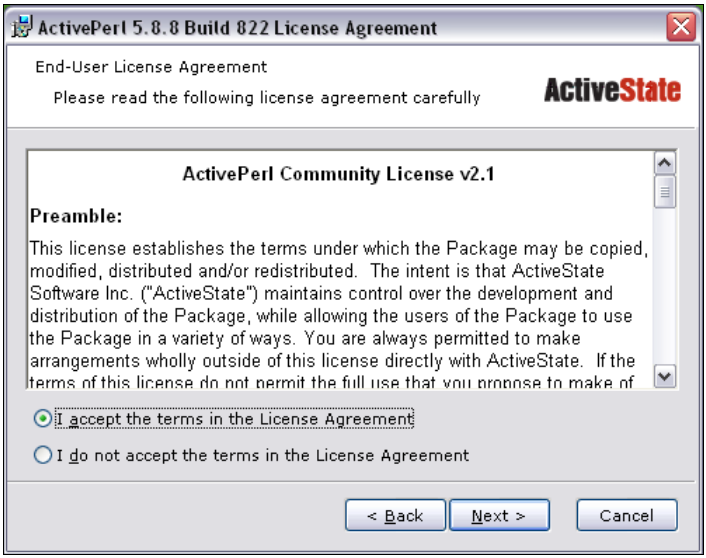


Рис. 1.23. Установка ActivePerl. Шаг 2

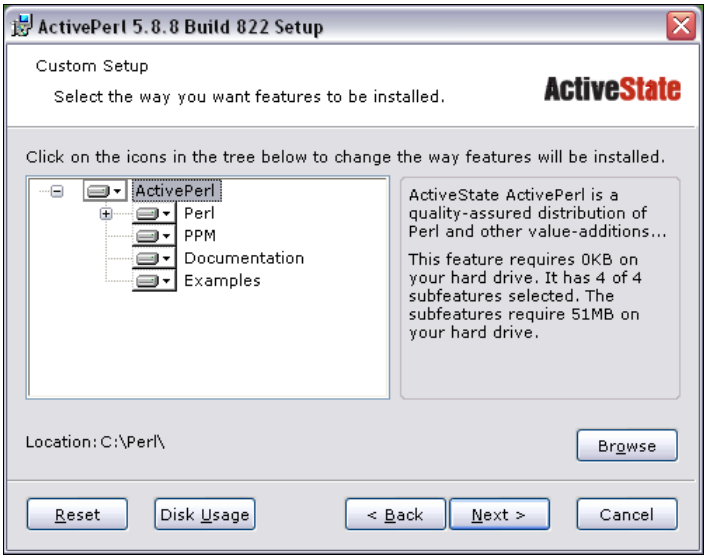


Рис. 1.24. Установка ActivePerl. Шаг 3

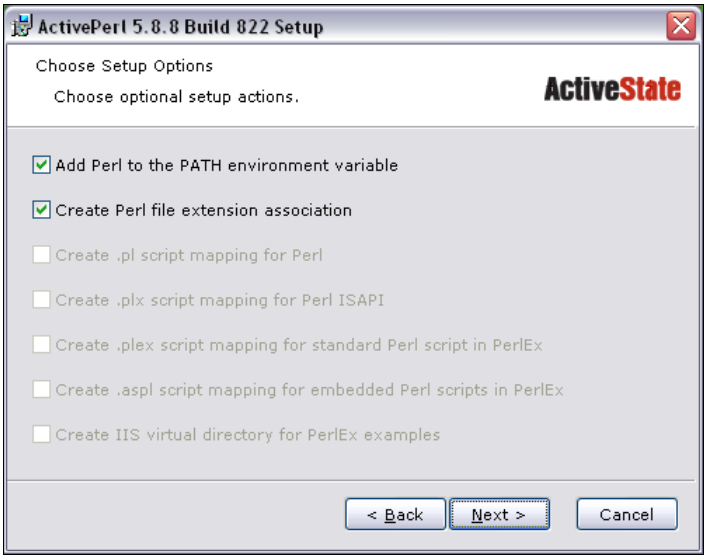


Рис. 1.25. Установка ActivePerl. Шаг 4

В открывшемся окне (рис. 1.26) нажимаем кнопку **Install** для начала установки.

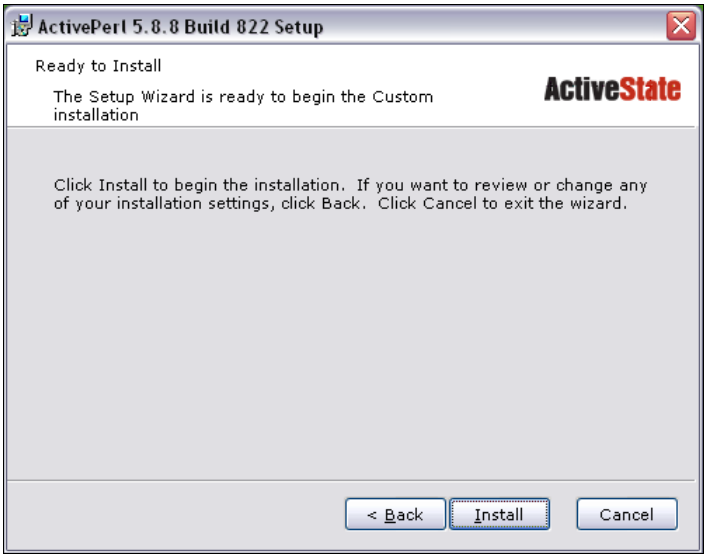


Рис. 1.26. Установка ActivePerl. Шаг 5

В следующем окне (рис. 1.27) нажимаем кнопку **Finish** для завершения процесса установки.

Рис. 1.27. Установка ActivePerl. Шаг 6

Для работы с MySQL из Perl необходимо установить модуль DBD-mysql. Для установки выбираем пункт меню **Пуск | Выполнить...** и в окне **Запуск программы** в поле **Открыть** набираем `cmd`, а затем нажимаем кнопку **ОК**. В командной строке набираем команду:

```
ppm install http://theoryx5.uwinnipeg.ca/ppms/DBD-mysql.ppd
```

Процесс установки модуля изображен на рис. 1.28.

Для того чтобы не менять путь после настройки программы на локальном компьютере, в конфигурационный файл сервера Apache (httpd.conf) необходимо добавить строку:

```
ScriptInterpreterSource registry
```

В этом случае сервер Apache игнорирует путь указанный в первой строке программы и производит поиск интерпретатора в реестре Windows. По этой причине мы можем указать путь к интерпретатору на сервере хостинг-провайдера, а тестировать программу на локальном компьютере. В противном случае пришлось бы прописывать путь на Windows-машине:

```
#!c:\perl\bin\perl.exe
```

А перед загрузкой программы на сервер необходимо было бы изменять его:

```
#!/usr/bin/perl
```

Учитывая, что все программы в этой книге написаны под Денвер, после указания директивы `ScriptInterpreterSource` вы сможете без изменений тести-

ровать программы на основе листингов книги на своем компьютере. Сохраняем файл httpd.conf и перезапускаем сервер Apache.

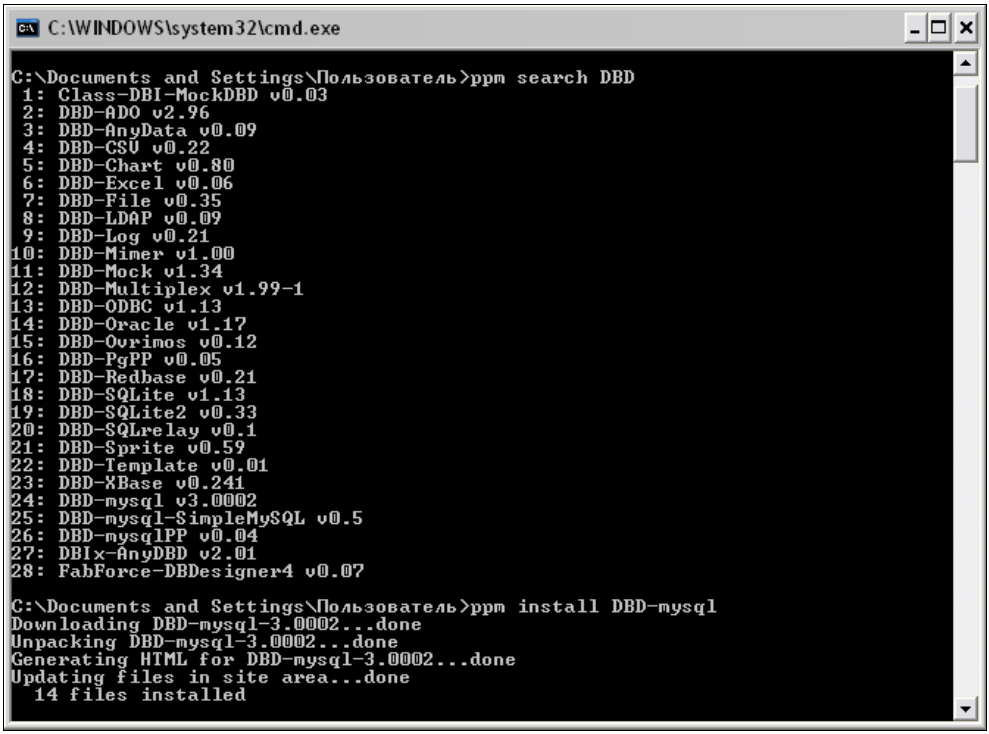
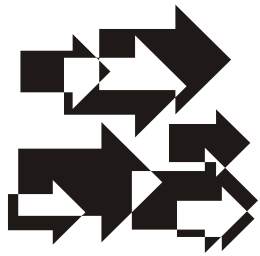


Рис. 1.28. Установка модуля DBD-mysql

ГЛАВА 2



Основы Perl

2.1. Основные понятия

Perl — это язык программирования на стороне сервера. В отличие от языка JavaScript, код Perl не зависит от программного обеспечения клиента и поэтому будет выполнен всегда.

Последовательность инструкций (называемая программой, сценарием или скриптом) выполняется *интерпретатором* языка Perl. Обработка Perl-кода производится на сервере до того, как страница будет передана Web-браузеру. В итоге Web-браузер получит обычный HTML-код или другой вывод.

Создать код программы позволяет любой текстовый редактор, например программа Блокнот в Windows. При использовании Блокнота следует помнить, что перевод строки в операционной системе Windows состоит из последовательности двух символов — `\r` (перевод каретки) и `\n` (перевод строки). На серверах хостинг-провайдеров обычно установлена операционная система семейства UNIX, например, FreeBSD. В этой операционной системе перевод строки осуществляется только одним символом `\n`. Если загрузить файл программы по протоколу FTP в бинарном режиме, то символ `\r` вызовет фатальную ошибку. По этой причине файлы по протоколу FTP следует загружать только в текстовом режиме (режим `ASCII`). В этом режиме символ `\r` будет удален автоматически. После загрузки файла следует установить права на выполнение. Для исполнения скриптов на Perl устанавливаем права в 755 (`-rwxr-xr-x`). Изменить права доступа позволяет практически любой FTP-клиент. Более подробно права доступа к файлам и работу с FTP-клиентами мы рассмотрим в соответствующих разделах книги.

2.1.1. Первая программа на Perl

При изучении языков программирования принято начинать с программы, выводящей надпись "Hello, world". Не будем нарушать традицию и продемонстрируем, как это будет выглядеть на Perl. Для начала рассмотрим возможность запуска программ из командной строки. Открываем Блокнот и набираем код из листинга 2.1.

Листинг 2.1. Первая программа для выполнения из командной строки

```
#!/usr/bin/perl -w
print "Hello, world";
```

Сохраняем в формате pl (например, index.pl) в папке C:\WebServers. Запускаем программу cmd.exe. Для этого выбираем пункт **Пуск | Выполнить...** и в окне **Запуск программы** в поле **Открыть** набираем `cmd`, а затем нажимаем кнопку **OK**. Для смены текущей папки в командной строке набираем команду:

```
cd C:\WebServers
```

В командной строке должно отобразиться приглашение:

```
C:\WebServers>
```

Для запуска нашей программы в командной строке набираем команду:

```
perl index.pl
```

Нажимаем клавишу <Enter>. В итоге отобразится надпись "Hello, world". Запускать Денвер нет необходимости, при установке мы добавили путь с интерпретатором Perl в переменную `PATH`.

Теперь рассмотрим программу вывода надписи в окне Web-браузера. Для этого открываем Блокнот и набираем код из листинга 2.2.

Листинг 2.2. Первая программа для вывода надписи в окне Web-браузера

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<HTML>";
print "<HEAD>";
print "<TITLE>Первая программа</TITLE>";
print "</HEAD>";
print "<BODY>";
```

```
print "Hello, world";
print "</BODY>";
print "</HTML>";
```

Сохраняем в формате pl (например, index.pl) в папке C:\WebServers \home\perlbook.ru\cgi-bin. Запускаем Денвер, а затем открываем Web-браузер и в адресной строке набираем адрес:

```
http://perlbook.ru/cgi-bin/index.pl
```

В итоге отобразится надпись "Hello, world". Теперь давайте отобразим исходный HTML-код:

```
<HTML><HEAD><TITLE>Первая программа</TITLE></HEAD><BODY>Hello,
world</BODY></HTML>
```

Как не трудно заметить, никаких признаков Perl в исходном коде нет. При выводе HTML-тегов с помощью оператора `print` весь код отображается на одной строке. Чтобы отобразить каждый тег на отдельной строке, необходимо добавить символ перевода строки (листинг 2.3). Для системы UNIX таким символом будет `\n`. В операционной системе Windows перевод строки состоит из комбинации двух символов `\r\n`.

Листинг 2.3. Вывод каждого тега на отдельной строке

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<HTML>\n";
print "<HEAD>\n";
print "<TITLE>Первая программа</TITLE>\n";
print "</HEAD>\n";
print "<BODY>\n";
print "Hello, world\n";
print "</BODY>\n";
print "</HTML>";
```

Теперь каждый тег будет на своей строке (листинг 2.4).

Листинг 2.4. Результат вывода предыдущей программы

```
<HTML>
<HEAD>
```

```
<TITLE>Первая программа</TITLE>
</HEAD>
<BODY>
Hello, world
</BODY>
</HTML>
```

При выводе HTML-тегов с помощью оператора `print` следует помнить, что теги могут иметь параметры, значения которых заключаются в кавычки. Например, если попробовать вывести так, как показано в листинге 2.5, то возникнет ошибка:

```
syntax error at Z:/home/perlbook.ru/cgi-bin/index.pl line 10, near
""<FONT color="red"
Execution of Z:/home/perlbook.ru/cgi-bin/index.pl aborted due to
compilation errors.
```

Листинг 2.5. Ошибочный код при выводе кавычек

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<HTML><HEAD>\n";
print "<TITLE>Первая программа</TITLE>\n";
print "</HEAD><BODY>\n";
print "<FONT color="red">\n"; # Строка с ошибкой
print "Hello, world\n";
print "</FONT>\n";
print "</BODY></HTML>\n";
```

Обойти данную проблему можно следующими способами:

- добавить защитный слэш перед каждой кавычкой:

  ```
  print "<FONT color=\"red\">\n";
  ```

- в операторе `print` использовать не кавычки, а апострофы. При использовании этого способа могут возникнуть проблемы. Например, в этом случае нельзя использовать специальные символы (`\n`). Кроме того, если внутри используется переменная, то вместо ее значения мы увидим имя переменной:

  ```
  print '<FONT color="red">';
  ```

ОБРАТИТЕ ВНИМАНИЕ

Все выражения в Perl заканчиваются точкой с запятой. Отсутствие точки с запятой в Perl приведет к остановке выполнения сценария и генерации сообщения об ошибке. Это самая распространенная ошибка для начинающих изучать язык Perl.

2.1.2. Структура программы

Давайте разберем каждую строку программы. В первой строке указывается путь к интерпретатору Perl:

```
#!/usr/bin/perl
```

Эта строка обязательна для операционной системы семейства UNIX. На некоторых серверах путь к интерпретатору выглядит по-другому:

```
#!/usr/local/bin/perl
```

Прежде чем закачивать скрипты на сервер хостинг-провайдера, следует узнать месторасположение интерпретатора Perl. Обычно эту информацию можно найти в разделе "Инструкции" или "FAQ".

После пути мы указываем флаг `-w`. Это позволяет выводить предупреждающие сообщения при компиляции программы:

```
#!/usr/bin/perl -w
```

Следующие строки являются комментариями:

```
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

Все, что указано после символа `#` до конца строки, игнорируется интерпретатором. Это позволяет вставлять в текст программы примечания, которые в дальнейшем помогут вспомнить назначение блока кода. Исключение составляет последовательность `#!`. Она используется для указания пути к интерпретатору.

Все сообщения об ошибках обычно записываются в журнал ошибок (error.log). Логи сервера можно найти с папке C:\WebServers\usr\local \apache\logs. Просмотреть содержимое журнала позволяет любой текстовый редактор, например Блокнот. Но каждый раз просматривать этот журнал не очень удобно. Для вывода сообщений об ошибках в окно Web-браузера мы указываем в программе следующую строку:

```
use CGI::Carp qw(fatalsToBrowser);
```

С помощью оператора `use` подключаем модуль `CGI::Carp`. Теперь при возникновении ошибки можно сразу увидеть ее описание.

Обратите внимание

В описании ошибки обычно указывается номер строки, в которой содержится ошибка. Для ее исправления достаточно отсчитать указанное количество строк и исправить ошибку. Если указана последняя строка, то, скорее всего, отсутствует закрывающая скобка в любом месте программы.

В следующей строке с помощью оператора `print` мы указываем заголовки ответа сервера:

```
print "Content-type: text/html\n\n";
```

В данном случае указывается тип документа. На данном этапе достаточно вставлять эту строку во все создаваемые программы. Если не включить эту строку, то будет выведено сообщение об ошибке.

Обратите внимание

Заголовки отделяются от основного содержания с помощью двух символов перевода строки (`\n\n`).

Все остальное содержимое документа выводится в Web-браузер с помощью оператора `print`:

```
print "Hello, world";
```

2.1.3. Комментарии в Perl-сценариях

Все, что расположено после символа `#` до конца строки, в Perl считается *однострочным комментарием*. Исключение составляет последовательность `#!`. Она используется для указания пути к интерпретатору.

```
# Однострочный комментарий
```

Однострочный комментарий можно записать после выражения:

```
print "Hello, world"; # Однострочный комментарий
```

Однострочные комментарии позволяют вставлять примечания в текст программы. В дальнейшем они помогут вспомнить назначение какого-либо блока кода.

2.1.4. Вывод результатов работы скрипта

Вывести результат можно с помощью оператора `print` несколькими способами:

```
print "Hello, world<BR>";
print ("Hello, world<BR>");
print "Hello, ", "world", "<BR>";
print "Hello, world" . "<BR>";
print "Hello, ";
```

```
print "world";
print "<BR>";
```

В итоге в окно Web-браузера будет выведено пять строк с надписью "Hello, world". Для вывода больших объемов HTML-кода можно воспользоваться следующим синтаксисом:

```
print <<METKA1;
<Текст>
METKA1
```

После первой метки `METKA1` обязательно указывается точка с запятой. Далее идет блок текста, а затем указывается название метки. Название второй метки обязательно должно начинаться с начала строки. Кроме названия метки на строке не должно быть никаких других символов, в том числе и пробелов. Точка с запятой не указывается. Регистр в названии метки имеет значение. В названии метки не должно быть русских букв. Внутри блока можно вывести значение переменной. Пример вывода большого объема текста приведен в листинге 2.6.

Листинг 2.6. Вывод больших блоков текста

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print <<METKA1;
<HTML><HEAD>
<TITLE>Первая программа</TITLE>
</HEAD><BODY>
<FONT color="red">
Hello, world
</FONT>
</BODY></HTML>
METKA1
# конец кода
```

ОБРАТИТЕ ВНИМАНИЕ

После второй метки обязательно должен быть символ перевода строки. Чтобы показать это, мы вставили после метки комментарий. В противном случае получим сообщение об ошибке:

```
Can't find string terminator "METKA1" anywhere before EOF at
Z:/home/perlbook.ru/cgi-bin/index.pl line 7.
```

Если первую метку заключить в апострофы, то внутри блока будет выведено не значение переменной, а ее имя (листинг 2.7).

Листинг 2.7. Вывод переменных внутри блоков текста

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

$str = "Значение переменной";

print <<METKA1;
Метка без апострофов<BR>
$str
<BR><BR>
METKA1

print <<'METKA2';
Метка с апострофами<BR>
$str
METKA2
# Конец программы
```

Итак, в первом случае мы увидим значение переменной `$str`, а во втором случае — имя переменной.

2.1.5. Завершение выполнения скрипта

Для досрочного завершения Perl-сценария используется оператор `exit`:

```
exit;
```

Предположим, наш сайт содержит четыре страницы: index.pl (Главная страница), firma.pl (О фирме), price.pl (Продукция) и contact.pl (Контактная информация). Сделаем панель навигации для сайта. Переход на другие страницы будет осуществляться не с помощью ссылок, а путем выбора нужной страницы из списка. Для этого на всех страницах сайта должна присутствовать форма из листинга 2.8.

Листинг 2.8. Навигация по сайту с помощью списка

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

```
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<FORM action=\"go.pl\">\n";
print "<SELECT name=\"page\">\n";
print "<OPTION value=\"0\" selected>На главную\n";
print "<OPTION value=\"1\">О фирме\n";
print "<OPTION value=\"2\">Продукция\n";
print "<OPTION value=\"3\">Контакты\n";
print "</SELECT>\n";
print "<INPUT type=\"submit\" value=\"Go!\">\n";
print "</FORM>\n";
```

Далее создаем файл go.pl (листинг 2.9).

Листинг 2.9. Содержимое файла go.pl

```
#!/usr/bin/perl -w
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
my $page = param("page");

if (defined($page)) {
   if ($page == 1) {
      print "Location: firma.pl\n\n";
      exit;
   }
   elsif ($page == 2) {
      print "Location: price.pl\n\n";
      exit;
   }
   elsif ($page == 3) {
      print "Location: contact.pl\n\n";
      exit;
   }
   else {
      print "Location: index.pl\n\n";
      exit;
   }
}
else {
   print "Location: index.pl\n\n";
   exit;
}
```

Теперь при выборе страницы из списка и нажатии кнопки **Go!** мы попадем на нужную страницу. Мы использовали оператор `exit`, т. к. после перехода на нужную страницу выполнение остального кода просто не имеет смысла.

> ***Внимание!***
>
> Кроме указанного кода в файле go.pl не должно быть никаких операторов вывода.

2.1.6. Засыпание сценария

Функция `sleep()` прерывает выполнение сценария на указанное время. По истечении срока сценарий продолжит работу:

```
sleep(<Время в секундах>);
```

Для примера: выведем начальное время, затем прервем выполнение скрипта на 30 секунд. По истечении срока выведем конечное время. Исходный код программы приведен в листинге 2.10.

Листинг 2.10. Засыпание сценария

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "Начало: ", time(), "<BR>";
sleep(30); # Засыпание сценария на 30 секунд
print "Конец: ", time();
```

Вывод:

```
Начало: 1218556140
Конец: 1218556170
```

2.1.7. Специальные символы

Специальные символы — это комбинации знаков, обозначающих служебные или непечатаемые символы, которые невозможно вставить обычным способом. Перечислим специальные символы, доступные в Perl:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\f` — перевод страницы;
- `\t` — знак табуляции;

- \' — апостроф;
- \" — кавычка;
- \$ — знак доллара;
- \% — знак процента;
- \@ — символ @;
- \\ — обратный слэш.

2.2. Переменные

Переменные — это участки памяти, используемые программой для хранения данных. Каждая переменная должна иметь уникальное имя в программе, состоящее из латинских букв, цифр и знаков подчеркивания. Все имена скалярных переменных в Perl начинаются со знака $. После знака $ нельзя указывать цифру.

Правильные имена скалярных переменных:

```
$x, $strName, $y1, $_name
```

Неправильные имена скалярных переменных:

```
y, ИмяПеременной, $1y, $ИмяПеременной
```

Последнее имя неправильное из-за русских букв. При указании имени переменной важно учитывать регистр букв:

```
$strName и $strname — разные переменные.
```

В качестве имени переменной не стоит использовать $a и $b: в некоторых функциях (например, sort()) они имеют специальное значение.

2.2.1. Типы данных и инициализация переменных

В Perl переменные могут содержать следующие типы данных:

- *скаляр* — целые числа, вещественные числа, строки, ссылки;
- *массив* — массив скаляров;
- *хеш* — ассоциативный массив.

Интерпретатор относит переменную к определенному типу по первому символу:

- все имена скалярных переменных начинаются с символа $;
- для массивов указывается символ @;
- имена хешей начинаются с символа %.

При инициализации переменной интерпретатор автоматически относит переменную к одному из типов данных. Значение переменной присваивается с помощью оператора =:

```
$number=7; # целое число
$number2=7.8; # вещественное число
$string="Строка"; # Переменной $string присвоено значение Строка
$string2='Строка'; # Переменной $string2 также присвоено значение Строка
```

Perl в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней:

```
$var="Строка"; # строка
$var=7; # целое число
```

В отличие от большинства языков программирования, в языке Perl нет способа определить, какой тип имеет скалярная переменная (целое число, вещественное число или строка).

Если в переменной нужно сохранить большой объем текста, то можно воспользоваться кодом из листинга 2.11.

Листинг 2.11. Запись в переменную объемного текста

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

$Y=<<Metka1;
<HTML>
<HEAD>
<TITLE>Строки</TITLE>
</HEAD>
<BODY>
Metka1
print $Y;
print "Привет";
print "</BODY>";
print "</HTML>";
```

В данном примере многострочный текст располагается между метками (`Metka1`):

```
$Y=<<Metka1;
...
Metka1
```

После первой метки `Metka1` обязательно указывается точка с запятой. Далее идет блок текста, а затем указывается название метки. Вторая метка обязательно должна начинаться с новой строки. Кроме названия метки, в строке не должно быть никаких других символов, в том числе и пробелов. Точка с запятой не указывается. Регистр в названии метки имеет значение. В названии метки не должно быть русских букв.

> ***Обратите внимание***
>
> После второй метки обязательно должен быть символ перевода строки. В противном случае получим сообщение об ошибке.

Если первую метку заключить в апострофы, то внутри блока будет выведено не значение переменной, а ее имя (листинг 2.12).

Листинг 2.12. Вывод переменных внутри блоков текста

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

$str = "Значение переменной";

$text1=<<METKA1;
Метка без апострофов<BR>
$str
<BR><BR>
METKA1
print $text1;

$text2=<<'METKA2';
Метка с апострофами<BR>
$str
METKA2
print $text2;
```

Вывод, соответствующий листингу 2.12:

```
Метка без апострофов
Значение переменной

Метка с апострофами
$str
```

Итак, в первом случае мы видим значение переменной `$str`, а во втором случае — имя переменной.

2.2.2. Проверка существования переменной

С помощью функции `defined(<Переменная>)` можно проверить существование переменной. Если переменной присвоено значение, то возвращается `true`. Если переменная объявлена, но ей не присвоено значение, то функция возвращает `false`. Для примера: переделаем нашу первую программу так, чтобы программа здоровалась не со всем миром, а только с нами (листинг 2.13).

Листинг 2.13. Проверка существования переменной

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$name = param("name");
print "<HTML><HEAD>\n";
print "<TITLE>Первая программа</TITLE>\n";
print "</HEAD><BODY>\n";
if (defined($name)) {
   print "Hello, $name";
}
else {
   print "Введите ваше имя<BR>\n";
   print "<FORM>\n";
   print "<INPUT type=\"text\" name=\"name\">\n";
   print "<INPUT type=\"submit\" value=\"OK\">\n";
   print "</FORM>\n";
}
print "</BODY></HTML>\n";
```

При первом запуске программы появится приглашение ввести имя. Вводим свое имя (например, `Николай`) и нажимаем **OK**. В итоге отобразится приветствие:

```
Hello, Николай
```

Для обработки данных формы мы подключаем модуль CGI с помощью оператора use:

```
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
```

При помощи qw(:standard) мы можем обратиться к значению элемента формы, просто указав его имя в функции param(). Если эту строку не указать, то к функции param() нужно обратиться следующим образом:

```
# подключаем модуль для обработки данных формы
use CGI;
print "Content-type: text/html\n\n";
$s = new CGI;
$name = $s->param("name");
```

Далее мы с помощью функции defined() проверяем существование переменной $name. Если переменная не определена, то выводим форму для ввода имени. Если переменная определена, то выводим приветствие с помощью оператора print:

```
   print "Hello, $name";
```

Теперь рассмотрим возможность ввода параметра с помощью командной строки. Открываем Блокнот и набираем код из листинга 2.14.

Листинг 2.14. Ввод параметра с помощью командной строки

```
#!/usr/bin/perl -w
print "Enter name:\n";
$name = <STDIN>;
print "Hello, $name";
```

Сохраняем в формате pl (например, index.pl) в папке C:\WebServers. Запускаем программу cmd.exe. Для этого выбираем пункт **Пуск | Выполнить...** и в окне **Запуск программы** в поле **Открыть** набираем cmd, а затем нажимаем **ОК**. Для запуска нашей программы в командной строке набираем команду:

```
perl C:\WebServers\index.pl
```

Нажимаем клавишу <Enter>. В итоге отобразится подсказка **Enter name:**. Вводим имя и нажимаем клавишу <Enter>. В итоге отобразится приветствие. При вводе русских букв возможны проблемы с кодировкой. По этой причине вводите имя латинскими буквами.

Рассмотрим программу построчно. В первой строке как обычно указывается путь к интерпретатору. Во второй строке мы выводим подсказку **Enter name:**. Это сообщение позволит определить, что конкретно требуется от

пользователя. Выражение <STDIN> дает возможность пользователю ввести значение с клавиатуры. В этой строке программа ожидает действия до момента нажатия клавиши <Enter>. После нажатия клавиши <Enter> введенное значение присваивается переменной $name, а затем с помощью оператора print приветствие выводится на экран монитора, и программа завершается.

Чтобы вывести русские буквы, можно воспользоваться следующим кодом:

```
#!/usr/bin/perl -w
use Encode;
print "Enter name:\n";
$name = <STDIN>;
$name = encode("iso-8859-1", $name);
print "Hello, $name";
```

Предварительно следует установить модуль Encode. Для этого запускаем Денвер, переходим на диск Z и запускаем файл Z:\usr\local\perl\bin\ppm-shell.bat. В командной строке должно быть приглашение:

```
ppm>
```

Набираем команду:

```
install Encode
```

Нажимаем клавишу <Enter>.

> ***ОБРАТИТЕ ВНИМАНИЕ***
> При установке модуля компьютер должен быть подключен к Интернету.

Для проверки существования переменной можно просто указать имя переменной в операторе if (листинг 2.15).

Листинг 2.15. Проверка переменной на значение

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

if (defined($var)) {
   print "$var Переменная определена";
}
else {
   print "Переменная не определена\n";
}
```

```
print "<BR>";
if ($var) {
   print "Переменная определена";
}
else {
   print "Переменная не определена\n";
}
```

В данном примере в обоих случаях будет выведено сообщение "Переменная не определена", т. к. переменная даже не объявлена в программе. Если переменной назначена пустая строка или 0, то в первом случае вернется "Переменная определена", а вот второй случай вернет "Переменная не определена". Следует с осторожностью применять данный метод, иначе это может привести к логическим ошибкам, которые очень трудно найти.

Для проверки существования элемента в хеше существует функция `exists()` (листинг 2.16).

Листинг 2.16. Проверка существования элемента в хеше

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
# Заполняем хеш значениями
%var = ('январь' => 1, 'февраль' => 2);
if (exists($var{'январь'})) {
   print "Переменная \$var{'январь'}определена";
}
else {
   print "Переменная \$var{'январь'} не определена\n";
}
print "<BR>";
if (exists($var{'март'})) {
   print "Переменная \$var{'март'} определена";
}
else {
   print "Переменная \$var{'март'} не определена\n";
}
```

Вывод:

```
Переменная $var{'январь'} определена
Переменная $var{'март'} не определена
```

В первом случае ключ январь определен в хеше %var, а во втором — ключа март нет в хеше %var. Более подробно хеши мы рассмотрим далее.

2.2.3. Преобразование типов данных

Что будет, если к числу прибавить строку? Например:

```
$Str = "5"; # Строка
$Number = 3; # Число
$Str2 = $Number + $Str; # Переменная содержит число 8
$Str3 = $Str + $Number; # Переменная содержит число 8
```

Результат будет абсолютно другим, нежели в JavaScript, оператор + в Perl не используется для конкатенации строк. В этом случае интерпретатор попытается преобразовать переменные к одному типу данных, а затем выполнить операцию. В нашем случае переменная $Str, тип строка, будет преобразована к числовому типу, а затем будет произведена операция сложения двух чисел.

Но что будет, если строку невозможно преобразовать в число? Скажем:

```
$Str = "Привет"; # Строка
$Number = 3; # Число
$Str2 = $Number + $Str; # Переменная содержит число 3
$Str3 = $Str + $Number; # Переменная содержит число 3
```

А что будет, если из числа вычесть строку, число умножить на строку или число разделить на строку? Рассмотрим примеры:

```
$Number = 15;
$Str = "5";
$Str2 = $Number - $Str; # Переменная содержит число 10
$Str3 = $Number * $Str; # Переменная содержит число 75
$Str4 = $Number / $Str; # Переменная содержит число 3
```

Итак, интерпретатор попытается преобразовать строку в число, а затем вычислить выражение. Причем не важно, в какой последовательности будут указаны число и строка:

```
$Str5 = $Str * $Number; # Переменная все равно содержит число 75
```

Но что будет, если в строке будут одни буквы? Возьмем:

```
$Number = 15;
$Str = "Строка";
$Str2 = $Number - $Str; # Переменная содержит число 15
$Str3 = $Str - $Number; # Переменная содержит число -15
$Str4 = $Number * $Str; # Переменная содержит число 0
$Str5 = $Str * $Number; # Переменная содержит число 0
```

```
$Str6 = $Number / $Str; # Ошибка деления на 0
$Str7 = $Str / $Number; # Переменная содержит число 0
```

С одной стороны, хорошо, что интерпретатор делает преобразование типов данных за нас. Но можно получить результат, который вовсе не планировался. По этой причине лучше оперировать переменными одного типа.

2.2.4. Удаление значения переменной

Удалить значение переменной и сделать ее неопределенной можно с помощью функции `undef()` (листинг 2.17).

Листинг 2.17. Удаление значения переменной

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

$var = "Значение переменной";
if (defined($var)) {
   print "Переменная определена";
}
else {
   print "Переменная не определена\n";
}
print "<BR>После функции undef<BR>\n";
undef $var;
if (defined($var)) {
   print "Переменная определена";
}
else {
   print "Переменная не определена\n";
}
```

Вывод:

```
Переменная определена
После функции undef
Переменная не определена
```

Это необходимо, если переменная использовалась при обработке данных большого объема и больше не нужна. Применение функции `undef()` позволит освободить память компьютера.

Функция `delete()` позволяет удалить элемент из хеша (листинг 2.18).

Листинг 2.18. Удаление элемента из хеша

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
# Заполняем хеш значениями
%var = ('январь' => 1, 'февраль' => 2);
if (exists($var{'январь'})) {
   print "Переменная \$var{'январь'} определена";
}
else {
   print "Переменная \$var{'январь'} не определена\n";
}
print "<BR>";
# Удаляем элемент хеша
delete($var{'январь'});
if (exists($var{'январь'})) {
   print "Переменная \$var{'январь'} определена";
}
else {
   print "Переменная \$var{'январь'} не определена\n";
}
```

Вывод:

```
Переменная $var{'январь'} определена
Переменная $var{'январь'} не определена
```

Обратите внимание на вывод знака `$` в окно Web-браузера. Если перед знаком `$` не указать защитный слэш, то в результате будет выведено значение элемента хеша с ключом `январь`:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
# Заполняем хеш значениями
%var = ('январь' => 1, 'февраль' => 2);
print "Переменная \$var{'январь'} определена<BR>";
print "Переменная $var{'январь'} определена";
```

Вывод:

```
Переменная $var{'январь'} определена
Переменная 1 определена
```

2.2.5. Создание и использование констант

Константы используются для хранения значений, которые не должны изменяться во время работы программы. Создать константу можно с помощью прагмы (модуля) `constant` (листинг 2.19).

Листинг 2.19. Создание константы

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
# Создание константы
use constant HOST_CONNECT => 'localhost';
print HOST_CONNECT;
```

После объявления константы ее имя указывается в программе без знака `$`.

2.2.6. Специальные литералы

В языке Perl доступны следующие литералы:

- `__LINE__` — содержит номер текущей строки в программе;
- `__FILE__` — имя и путь к файлу с текущей программой;
- `__END__` — служит для указания логического конца программы. Фрагмент кода, расположенный после этого литерала, не обрабатывается интерпретатором языка;
- `__DATA__` — служит для указания логического конца программы и дополнительно открывает файл с дескриптором DATA для чтения информации после литерала.

Рассмотрим пример:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
```

```
print "Текущая строка — ", __LINE__, "<BR>";
print "Название файла — ", __FILE__, "<BR>";
__END__
print "Текст, расположенный после литерала __END__";
```

Вывод:

```
Текущая строка — 6
Название файла — Z:/home/perlbook.ru/cgi-bin/index.pl
```

Как видно из примера, мы не получили вывод оператора `print` из последней строки, т. к. он расположен после литерала `__END__`.

2.3. Операторы Perl

Операторы позволяют произвести определенные действия с данными. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы позволяют произвести арифметические вычисления, а оператор конкатенации строк используется для соединения двух строк в одну. Рассмотрим операторы, доступные в Perl, более подробно.

2.3.1. Математические операторы

Операторы, доступные в Perl:

- `+` — сложение:

```
$z = $x + $y;
```

- `-` — вычитание:

```
$z = $x — $y;
```

- `*` — умножение:

```
$z = $x * $y;
```

- `/` — деление:

```
$z = $x/$y;
```

- `%` — остаток от деления:

```
$z = $x%$y;
```

- `**` — возведение в степень:

```
$z = $x**$y;
```

- `++` — оператор инкремента. Увеличивает значение переменной на 1:

```
$z++; # Эквивалентно $z =$z + 1;
```

- `--` — оператор декремента. Уменьшает значение переменной на 1:

```
$z--; # Эквивалентно $z =$z – 1;
```

Операторы инкремента и декремента могут использоваться в постфиксной или префиксной формах:

```
$z++; $z--; # Постфиксная форма
++$z; --$z; # Префиксная форма
```

В чем разница? При постфиксной форме (`$z++`) возвращается значение переменной перед операцией, а при префиксной форме (`++$z`) сначала производится операция и только потом возвращается значение. Продемонстрируем это на примере (листинг 2.20).

Листинг 2.20. Постфиксная и префиксная форма

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<HTML><HEAD>\n";
print "<TITLE>Постфиксная и префиксная форма</TITLE>\n";
print "</HEAD><BODY>\n";
$x = 5;
$z = $x++; # $z = 5, $x = 6
print "<B>Постфиксная форма (\$z = \$x++;):</B><BR> ";
print "\$z = $z <BR>\$x = $x <BR><BR>";
$x = 5;
$z = ++$x; # $z = 6, $x = 6
print "<B>Префиксная форма (\$z=++\$x;):</B><BR> \$z = $z <BR>\$x = $x";
print "</BODY></HTML>\n";
```

В итоге получим следующий результат:

```
Постфиксная форма ($z = $x++;):
$z = 5
$x = 6

Префиксная форма ($z=++$x;):
$z = 6
$x = 6
```

2.3.2. Операторы присваивания

Операторы, доступные в Perl:

- = — присваивает переменной значение:

```
$z = 5;
```

- += — увеличивает значение переменной на указанную величину:

```
$z += 5; # Эквивалентно $z = $z + 5;
```

- -= — уменьшает значение переменной на указанную величину:

```
$z -= 5; # Эквивалентно $z = $z – 5;
```

- *= — умножает значение переменной на указанную величину:

```
$z *= 5; # Эквивалентно $z = $z*5;
```

- /= — делит значение переменной на указанную величину:

```
$z /= 5; # Эквивалентно $z = $z/5;
```

- %= — делит значение переменной на указанную величину и возвращает остаток:

```
$z %= 5; # Эквивалентно $z = $z%5;
```

- **= — возводит значение переменной в указанную степень:

```
$z **= 5; # Эквивалентно $z = $z**5;
```

2.3.3. Операторы обработки строк. Запуск внешних программ

Рассмотрим . — оператор конкатенации строк:

```
$Z = "Строка1" . "Строка2";
# Переменная $Z будет содержать значение "Строка1Строка2"
```

Очень часто необходимо сформировать строку, состоящую из имени переменной и ее значения. Если написать:

```
$X = "Строка1";
$Z = "Значение равно $X";
```

Переменная `$Z` будет содержать значение `"Значение равно Строка1"`, а если написать так:

```
$X = "Строка1";
$Z = 'Значение равно $X';
```

Переменная `$Z` будет содержать значение `"Значение равно $X"`. Помните, строка в кавычках и строка в апострофах вернут разный результат. В последнем случае, для того чтобы получить значение переменной, можно воспользоваться операцией конкатенации строк:

```
$X = "Строка1";
$Z = 'Значение равно ' . $X;
```

Рассмотрим еще один пример. Предположим, нужно объединить два слова в одно. Одно из слов задано с помощью переменной:

```
$X = "авто";
$Z = "$Xтранспорт"; # $Z = "автотранспорт"
```

Переменная `$Z` будет содержать значение `автотранспорт` только по причине того, что имя переменной не может содержать русских букв. Попробуем вывести так:

```
$X = "auto";
$Z = "$Xtransport"; # $Z = ""
```

Переменная `$Z` будет содержать пустую строку, т. к. переменная `$Xtransport` не определена. В этом случае лучше воспользоваться следующими способами:

- использовать конкатенацию строк:

```
$X = "auto";
$Z = $X . "transport"; # $Z = "autotransport"
```

- указать имя переменной в фигурных скобках:

```
$X = "auto";
$Z = "${X}transport"; # $Z = "autotransport"
```

При помощи оператора `x` можно размножить текущую строку:

```
$Str = "Привет" x 3;
print $Str; # выведет ПриветПриветПривет
```

Для строк можно также использовать оператор `++`. Обратите внимание: этот оператор работает только с латинскими буквами. Русские буквы сразу приравниваются к нулю:

```
$Str = "b";
$Str++;
print $Str, "<BR>"; # выведет c
++$Str;
print $Str, "<BR>"; # выведет d
```

Если содержимое строки заключить в обратные кавычки, то это позволит запустить внешнюю программу и присвоить переменной результат ее работы (листинг 2.21).

Листинг 2.21. Запуск внешней программы

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "<HTML><HEAD>\n";
print "<TITLE>Запуск внешних программ</TITLE>\n";
print "</HEAD><BODY>\n";
$X = `dir`;
print "<TEXTAREA cols=70 rows=30>";
print $X;
print "</TEXTAREA>";
print "</BODY></HTML>\n";
```

Данный код выведет содержимое папки Z:\home\perlbook.ru\cgi-bin. При выводе используется кодировка Dos (кодовая страница 866), поэтому русские буквы будут искажены.

2.3.4. Приоритет выполнения операторов

В какой последовательности будет вычисляться это выражение:

```
$X = 5 + 10 * 3 / 2;
```

Это зависит от приоритета выполнения операторов. В данном случае последовательность вычисления выражения будет следующей:

1. Число 10 будет умножено на 3, приоритет операции умножения выше приоритета операции сложения.
2. Полученное значение будет поделено на 2, приоритет операции деления равен приоритету операции умножения, но выше операции сложения.
3. К полученному значению будет прибавлено число 5, оператор присваивания = имеет наименьший приоритет.
4. Значение будет назначено переменной `$X`.

С помощью скобок можно изменить последовательность вычисления выражения:

```
$X = (5 + 10) * 3 / 2;
```

1. К числу 5 будет прибавлено 10.
2. Полученное значение будет умножено на 3.

3. Полученное значение будет поделено на 2.
4. Значение будет присвоено переменной `$X`.

Перечислим операторы в порядке убывания приоритета:

- `++`, `--` — инкремент, декремент;
- `**` — возведение в степень;
- `*`, `/`, `%` — умножение, деление, остаток от деления;
- `+`, — — сложение, вычитание;
- `=`, `+=`, `-=`, `*=`, `/=`, `%=` — присваивание.

2.4. Массивы и хеши

Массив — это нумерованный набор скалярных переменных. Переменная в массиве называется *элементом* массива, а ее позиция в массиве задается *индексом*. Индексация элементов массива начинается с 0, а не с 1. Это следует помнить. Общее количество элементов в массиве называется *размером* массива. Первым символом в названии массива всегда указывается символ `@`.

Основным отличием хешей от массивов является возможность обращения к элементу хеша не по числовому индексу, а по индексу, состоящему из строки. Индексы хеша называются *ключами*, а сами хеши часто называют *ассоциативными массивами*. Первым символом в названии хеша всегда указывается символ `%`.

2.4.1. Инициализация массива

Инициализация массива осуществляется следующими способами:

- поэлементно:

```
$Mass[0] = "Ноль";
$Mass[1] = "Один";
$Mass[2] = "Два";
$Mass[3] = "Три";
```

> **ОБРАТИТЕ ВНИМАНИЕ**
>
> Вместо символа `@` мы указали символ `$`. Это происходит потому, что отдельные элементы массива являются скалярными переменными. Перед элементами массива всегда должен быть символ `$`, а не `@`. К этому очень трудно привыкнуть начинающему программисту. По этой причине часто возникают ошибки.

- указав все элементы массива сразу:

```
@Mass = ("Ноль", "Один", "Два", "Три");
```

Обратите внимание

В этом случае мы указываем символ `@`, а не `$`.

- задав диапазон значений:

```
@Mass = (1..10);
print @Mass; # Выведет 12345678910
@Mass = ('a'..'z');
print @Mass; # Выведет abcdefghijklmnopqrstuvwxyz
```

- с помощью операции `qw`. В этом случае элементы указываются через пробел:

```
@Mass = qw(Ноль Один Два Три);
print @Mass; # Выведет НольОдинДваТри
```

2.4.2. Получение и изменение элементов массива. Определение последнего индекса массива

Обращение к элементам массива осуществляется с помощью квадратных скобок, в которых указывается индекс элемента. Индексация элементов массива начинается с нуля:

```
@Mass = ("Ноль", "Один", "Два", "Три");
$var = $Mass[1]; # Переменной $var будет присвоено значение "Один"
```

В языке Perl можно указывать в качестве индекса отрицательное число. В этом случае индекс будет отсчитываться от последнего элемента массива:

```
@Mass = ("Ноль", "Один", "Два", "Три");
$var = $Mass[-1]; # Переменной $var будет присвоено значение "Три"
```

С помощью списка можно присвоить значения элементов массива некоторым скалярным переменным:

```
@Mass = ("Ноль", "Один", "Два", "Три");
($var1, $var2, $var3, $var4) = @Mass;
print $var2; # Переменной $var2 будет присвоено значение "Один"
```

Для определения последнего индекса массива используется последовательность символов `$#` перед именем массива:

```
@Mass = ("Ноль", "Один", "Два", "Три");
print $#Mass; # Выведет 3
```

Данный пример выведет число 3, т. к. индексация массива начинается с 0. Чтобы получить количество элементов массива, необходимо к этому значению прибавить 1 или использовать инструкцию `scalar(@Mass)`:

```
@Mass = ("Ноль", "Один", "Два", "Три");
print scalar(@Mass); # Выведет 4
```

Если выражению `$#Mass` присвоить какое-либо значение, то это будет максимальный индекс массива. В этом случае если цифра больше максимального индекса текущего массива, то размер массива увеличится, а у новых элементов будет неопределенное значение. Если цифра меньше максимального индекса, то элементы с индексом больше указанного значения будут удалены:

```
@Mass = ("Ноль", "Один", "Два", "Три");
print $#Mass; # Выведет 3
$#Mass = 5;
print $#Mass; # Выведет 5
$#Mass = 2;
print @Mass; # Выведет НольОдинДва
```

При желании можно добавить новый элемент массива или изменить значение существующего:

```
$Mass[$#Mass+1]="Четыре";
$Mass[0]="Нуль";
```

Для добавления нового элемента в конец массива можно воспользоваться следующим способом:

```
@Mass = ("Ноль", "Один", "Два", "Три");
$Mass[@Mass]="Четыре";
```

2.4.3. Добавление и удаление элементов массива

Для добавления и удаления элементов массива используются следующие функции:

- `unshift(<Массив>, <Элемент>)` — добавляет элементы в начало массива:

```
$Mass[0]="Три";
$Mass[1]="Четыре";
unshift(@Mass, "Один", "Два");
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Один Два Три Четыре
```

- `push(<Массив>, <Элемент>)` — добавляет элементы в конец массива:

```
$Mass[0]="Один";
$Mass[1]="Два";
push(@Mass, "Три", "Четыре");
```

```
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Один Два Три Четыре
```

При желании можно добавить новый элемент в конец массива следующими способами:

```
@Mass = ("Ноль", "Один", "Два", "Три");
$Mass[$#Mass+1]="Четыре";
$Mass[@Mass]="Пять";
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Ноль Один Два Три Четыре Пять
```

- `shift(<Массив>)` — удаляет первый элемент массива и возвращает его:

```
$Mass[0]="Один";
$Mass[1]="Два";
print shift(@Mass); # Выведет "Один"
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Два
```

- `pop(<Массив>)` — удаляет последний элемент массива и возвращает его:

```
$Mass[0]="Один";
$Mass[1]="Два";
print pop(@Mass); # Выведет "Два"
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Один
```

С помощью функции `splice()` можно заменить часть массива одним элементом или массивом элементов. Функция имеет следующий формат:

```
splice(<Массив>, <Начальная позиция>, <Количество элементов>,
<Добавляемый массив>);
```

- `<Массив>` — исходный массив. Если остальные аргументы не указаны, то функция `splice()` удаляет все элементы из массива;
- `<Начальная позиция>` — количество элементов от начала массива, которые надо пропустить;
- `<Количество элементов>` — количество элементов, которое нужно заменить в исходном массиве. Если два последних аргумента не указаны, то будут удалены элементы от `<Начальной позиции>` до конца массива;
- `<Добавляемый массив>` — один элемент или массив элементов, добавляемых вместо выбранных элементов. Если аргумент не указан, то заданная секция удаляется из массива:

```
@Mass1=("Один", "Два", "Три", "Четыре", "Пять");
@Mass2=("3", "4", "5");
splice(@Mass1, 2, scalar(@Mass2), @Mass2);
for($i=0; $i<@Mass1; $i++) {
   print "$Mass1[$i] ";
} # Один Два 3 4 5
```

2.4.4. Переворачивание массива

Функция `reverse()` возвращает массив, элементы которого следуют в обратном порядке относительно исходного массива:

```
@Mass=("Один", "Два", "Три", "Четыре");
@Mass=reverse(@Mass);
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Четыре Три Два Один
```

2.4.5. Сортировка массива. Создание пользовательской сортировки

Функция `sort()` позволяет отсортировать массив в алфавитном порядке:

```
@Mass=("Один", "Два", "Три", "Четыре");
@Mass=sort @Mass;
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Два Один Три Четыре
```

Для создания пользовательской сортировки можно указать функцию для сравнения. Функция принимает две переменные `$a` и `$b` и должна возвращать:

- `1` — если первый больше второго;
- `-1` — если второй больше первого;
- `0` — если элементы равны.

Например, стандартная сортировка зависит от регистра символов:

```
@Mass=("единица1", "Единица2", "Единый");
@Mass=sort @Mass;
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Единица2 Единый единица1
```

В результате мы получим неправильную сортировку, ведь `"Единый"` и `"Единица2"` больше `"единица1"`. Изменим стандартную сортировку на свою сортировку без учета регистра (листинг 2.22).

Листинг 2.22. Сортировка без учета регистра

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');

print "Content-type: text/html\n\n";

@Mass=("единица1", "Единица2", "Единый");
@Mass=sort { f_sort($a, $b) } @Mass;
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # единица1 Единица2 Единый

# Функция пользовательской сортировки
sub f_sort {
   $str1 = lc($_[0]); # Преобразуем к нижнему регистру
   $str2 = lc($_[1]); # Преобразуем к нижнему регистру
   if ($str1 gt $str2) { return 1; }
   if ($str1 lt $str2) { return -1; }
   return 0;
}
```

Для этого две переменные приводим к одному регистру, а затем производим стандартное сравнение. Заметьте: мы не изменяем регистр самих элементов массива, а работаем с их копиями. Для правильной работы функции `lc()` с русским языком необходимо настроить *локаль*. Это позволяет сделать функция `setlocale()`. Более подробно мы рассмотрим функцию `setlocale()` при изучении функций обработки строк.

2.4.6. Перебор элементов массива

Для перебора массивов применяются три цикла — `for`, `while` и `foreach`:

- Цикл `for`:

```
@Mass=qw(Один Два Три Четыре);
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Один Два Три Четыре
```

- Цикл `while`:

```
@Mass=qw(Один Два Три Четыре);
$i=0;
while($i<@Mass) {
   print "$Mass[$i] ";
   $i++;
} # Один Два Три Четыре
```

- Цикл `foreach`:

```
@Mass=qw(Один Два Три Четыре);
foreach $var (@Mass) {
   print "$var ";
} # Один Два Три Четыре
```

Если не указать переменную, то при каждой итерации цикла специальной переменной `$_` будет присвоено значение элемента массива:

```
@Mass=qw(Один Два Три Четыре);
foreach (@Mass) {
   print "$_ ";
} # Один Два Три Четыре
```

2.4.7. Заполнение массива числами

Создать массив, состоящий из чисел, можно либо с помощью цикла, либо с помощью указания диапазона. Предположим, необходимо создать массив, состоящий из чисел от 1 до 100:

```
@Mass = (1 .. 100);
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i]<BR>";
}
```

2.4.8. Преобразование массива в строку

Функция `join()` преобразует массив в строку. Элементы добавляются через указанный разделитель:

```
@Mass = ("Фамилия", "Имя", "Отчество", "Год рождения");
$str = join(" — ", @Mass);
print "$str"; # $str = Фамилия — Имя — Отчество — Год рождения
```

2.4.9. Удаление оконечных пробельных символов

Рассмотрим функции удаления оконечных пробельных символов:

- `chomp()` — удаляет символ новой строки (`\n`) во всех элементах массива:

```
@Mass = ("Фамилия\n", "Имя\n", "Отчество\n", "Год рождения\n");
for($i=0; $i<@Mass; $i++) {
   print $Mass[$i];
}
   print "<BR>";
chomp(@Mass);
for($i=0; $i<@Mass; $i++) {
   print $Mass[$i];
}
```

Вывод в исходном HTML-коде:

```
Фамилия
Имя
Отчество
Год рождения
<BR>ФамилияИмяОтчествоГод рождения
```

- `chop()` — удаляет последние символы во всех элементах массива:

```
@Mass = ("Фамилия", "Имя", "Отчество", "Год рождения");
chop(@Mass);
$str = join(" — ", @Mass);
print "$str"; # $str = Фамили — Им — Отчеств — Год рождени
```

2.4.10. Функции *grep()* и *map()*

Функция `grep()` позволяет произвести поиск в массиве. Возвращает новый массив, элементы которого соответствуют условию. Функция имеет следующий формат:

```
<Новый массив> = grep(<Условие>, <Исходный массив>);
```

Предположим, есть массив, заполненный числами от 1 до 100. Создадим на его основе новый массив, в котором все элементы массива меньше или равны 50:

```
@Mass=(1..100);
@Mass2=grep($_<=50, @Mass);
for($i=0; $i<@Mass2; $i++) {
   print "$Mass2[$i]<BR>";
}
```

Функция `map()` позволяет применить функцию к каждому элементу массива и возвращает новый массив. Функция имеет следующий формат:

```
<Новый массив> = map(<Функция>, <Исходный массив>);
```

Предположим, есть массив, заполненный числами от 1 до 100. Создадим новый массив, в котором все элементы первого массива умножены на 3.

```
@Mass=(1..100);
@Mass2=map($_*3, @Mass);
for($i=0; $i<@Mass2; $i++) {
   print "$Mass2[$i]<BR>";
}
```

2.4.11. Хеши

Инициализация ассоциативного массива осуществляется следующими способами:

- поэлементно:

```
$Mass{"Один"} = 1;
$Mass{"Два"} = 2;
$Mass{"Три"} = 3;
print $Mass{"Один"}; # Выведет число 1
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Вместо квадратных скобок в ассоциативных массивах используются фигурные скобки.

- указав все элементы массива сразу:

```
%Mass = ("Один", 1, "Два", 2, "Три", 3);
print $Mass{"Два"}; # Выведет число 2
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В этом случае мы указываем символ `%`, а не `$` или `@`.

- с помощью операции `=>`. Это позволяет сделать процесс более наглядным:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
print $Mass{"Два"}; # Выведет число 2
```

Доступ к элементу ассоциативного массива осуществляется путем указания ключа в фигурных скобках:

```
print $Mass{"Два"};
```

С помощью функции `each()` можно присвоить значения пары ключ-значение некоторым скалярным переменным:

```
%Mass = ("Один" => 1);
($key, $value) = each(%Mass);
print "$key => $value"; # Выведет Один => 1
```

Цикл `while` позволяет вывести все пары ключ-значение:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
while (($key, $value) = each(%Mass)) {
   print "$key => $value ";
} # Выведет Два => 2 Три => 3 Один => 1
```

Функция `keys()` возвращает список всех ключей в хеше:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
@Mass2 = keys(%Mass);
print @Mass2; # Выведет ДваТриОдин
```

Если указать функцию `keys()` в цикле `foreach`, то можно перебрать все элементы хеша:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
foreach $var (keys(%Mass)) {
   print "$var => $Mass{$var} ";
} # Выведет Два => 2 Три => 3 Один => 1
```

Вместо цикла `foreach` можно применить цикл `while` в сочетании с функцией `pop()`:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
@Mass2 = keys(%Mass);
while ($var = pop(@Mass2)) {
   print "$var => $Mass{$var} ";
} # Выведет Один => 1 Три => 3 Два => 2
```

Функция `values()` возвращает список всех значений в хеше:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
@Mass2 = values(%Mass);
print @Mass2; # Выведет 231
```

Функция `delete()` позволяет удалить элемент из хеша:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
delete($Mass{'Один'});
print %Mass; # Выведет Два2Три3
```

С помощью функции `exists()` можно проверить существование элемента хеша:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
if (exists($Mass{'Один'})) {
   print "Переменная \$Mass{'Один'} определена";
}
else {
   print "Переменная \$Mass{'Один'} не определена\n";
} # Выведет Переменная $Mass{'Один'} определена
```

Ассоциативные массивы удобно использовать для подсчета вхождений слова в строку. Произведем подсчет вхождений и выведем результат в текущем порядке, по возрастанию и по убыванию количества вхождений (листинг 2.23).

Листинг 2.23. Подсчет вхождений слова в строку

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

@Mass=qw(два три два один три три один два три
                    три один два один один один);
for($i=0; $i<@Mass; $i++) {
   $Mass2{$Mass[$i]}++;
}
while (($key, $value) = each(%Mass2)) {
   print "$key => $value ";
} # Выведет один => 6 два => 4 три => 5
print "<BR>";
# Вывод с сортировкой по возрастанию
foreach (sort { $Mass2{$a} <=> $Mass2{$b} } keys(%Mass2)) {
   print "$_ => $Mass2{$_} ";
} # Выведет два => 4 три => 5 один => 6
print "<BR>";
# Вывод с сортировкой по убыванию
foreach (sort { $Mass2{$b} <=> $Mass2{$a} } keys(%Mass2)) {
   print "$_ => $Mass2{$_} ";
} # Выведет один => 6 три => 5 два => 4
```

2.5. Строки

В Интернете часто приходится производить манипуляции со строками. По этой причине необходимо знать и уметь использовать встроенные функции Perl, предназначенные для обработки строк. Например, перед добавлением

сообщения в гостевую книгу можно удалить лишние пробелы и все теги из строки, добавить защитные слэши перед специальными символами или заменить их на HTML-эквиваленты и т. д.

2.5.1. Операторы *q* и *qq*

В языке Perl существуют две формы представления строк:

- строка в апострофах;
- строка в кавычках.

Если строка задается в апострофах, то внутри строки специальные символы (например, `\n` или `\t`) теряют свое специальное значение:

```
$str = 'Значение1\nЗначение2';
print $str;
```

Вывод:

```
Значение1\nЗначение2
```

Если внутри строки в апострофах указать переменную, то вместо значения переменной мы получим только ее имя:

```
$var = 10;
$str = 'Значение равно $var';
print $str;
```

Вывод:

```
Значение равно $var
```

Для вывода символов `'` и `\` внутри строки в апострофах необходимо их экранировать с помощью защитного слэша:

```
$str = 'Д\'Артаньян и три мушкетера';
print $str;
```

Вывод:

```
Д'Артаньян и три мушкетера
```

Вместо одинарных кавычек можно использовать оператор `q`. Оператор имеет следующий формат:

```
q(<Строка>)
q/<Строка>/
q#<Строка>#
q[<Строка>]
```

При использовании этого оператора нет необходимости экранировать апостроф внутри строки:

```
$str = q(Д'Артаньян и три мушкетера);
print $str;
```

Вывод:

```
Д'Артаньян и три мушкетера
```

Если строка задается в кавычках, то символы \n, \t и др. имеют специальное значение. Например, символ \n соответствует переводу строки:

```
$str = "Значение1\nЗначение2";
print $str;
```

Вывод в исходном HTML-коде:

```
Значение1
Значение2
```

Если внутри строки в двойных кавычках указать переменную, то мы получим значение переменной:

```
$var = 10;
$str = "Значение равно $var";
print $str;
```

Вывод:

```
Значение равно 10
```

Для вывода символов ", \, $, @ и % внутри строки в двойных кавычках необходимо их экранировать с помощью защитного слэша:

```
$str = "Группа \"Кино\"";
print $str;
```

Вывод:

```
Группа "Кино"
```

Вместо двойных кавычек можно использовать оператор qq. Оператор имеет следующий формат:

```
qq(<Строка>)
qq/<Строка>/
qq#<Строка>#
qq[<Строка>]
```

При использовании этого оператора нет необходимости экранировать кавычки внутри строки:

```
$str = qq(Группа "Кино");
print $str;
```

Вывод:

```
Группа "Кино"
```

Что же лучше использовать, строку в апострофах или строку в кавычках? Как уже говорилось, строка в апострофах выводится "как есть". Интерпретатор не производит разбор строки в поисках переменных, и по этой причине обработка строк в апострофах производится быстрее. Если внутри строки необходимо вывести специальные символы или значения каких-либо переменных, то следует использовать кавычки. Можно также с помощью конкатенации строк комбинировать строку в апострофах со строкой в кавычках.

2.5.2. Функции для работы со строками

Перечислим основные функции для работы со строками:

- `length()` — возвращает количество символов в строке:

```
$str1 = "Строка\n";
print length($str1); # Выведет 7
print "<BR>";
$str2 = 'Строка\n';
print length($str2); # Выведет 8
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Если в строке содержатся специальные символы, то длина строки в кавычках будет отличаться от длины строки в апострофах.

- `chop()` — удаляет последний символ в конце строки и возвращает его:

```
$str1 = "Строка";
print "Из строки \"$str1\" <BR>";
$str2 = chop($str1);
print "мы удалили символ \"$str2\"<BR>";
print "в итоге строка выглядит так \"$str1\"";
```

Вывод:

```
Из строки "Строка"
мы удалили символ "а"
в итоге строка выглядит так "Строк"
```

Функция `chop()` работает также с массивами. В этом случае будут удалены последние символы во всех элементах массива:

```
@Mass = ("Фамилия", "Имя", "Отчество", "Год рождения");
chop(@Mass);
$str = join(" — ", @Mass);
print "$str"; # $str = Фамили — Им — Отчеств — Год рождени
```

- `chomp()` — удаляет символ новой строки (`\n`). Возвращает количество удаленных символов:

```
$str = "Строка\n\n\n\n";
$var = chomp($str);
print "Удален(о) $var символ(а)";
# Выведет Удален(о) 1 символ(а)
```

Обратите внимание, функция `chomp()` удалила только один символ новой строки из четырех. Для того чтобы удалить все четыре символа, необходимо присвоить пустую строку специальной переменной — `$/`:

```
$str = "Строка\n\n\n\n";
$/ = "";
$var = chomp($str);
print "Удален(о) $var символ(а)";
# Выведет Удален(о) 4 символ(а)
```

Переменной `$/` можно присвоить другое значение. Например, чтобы удалить символ табуляции, переменной необходимо присвоить значение `\t`:

```
$str = "Строка\t";
$/ = "\t";
$var = chomp($str);
print "Удален(о) $var символ(а)";
# Выведет Удален(о) 1 символ(а)
```

Функция `chomp()` работает также с массивами. В этом случае будет удален символ новой строки (`\n`) во всех элементах массива:

```
@Mass = ("Фамилия\n", "Имя\n", "Отчество\n", "Год рождения\n");
for($i=0; $i<@Mass; $i++) {
   print $Mass[$i];
}
print "<BR>";
chomp(@Mass);
for($i=0; $i<@Mass; $i++) {
   print $Mass[$i];
}
```

Вывод в исходном HTML-коде:

```
Фамилия
Имя
Отчество
Год рождения
<BR>ФамилияИмяОтчествоГод рождения
```

- `substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. Функция имеет следующий формат:

```
substr(<Строка>, <Начальная позиция>, [<Длина>], [<Строка для замены>]);
```

```
$str = "Строка";
$str1 = substr($str, 0, 1);
print "$str1"; # Выведет "С"
print "<BR>";
$str2 = substr($str, 1);
print "$str2"; # Выведет "трока"
```

Если указана строка для замены, то указанный фрагмент будет заменен в исходной строке:

```
$str = "Строка";
substr($str, 4, 2, "йка");
print "$str"; # Выведет "Стройка"
```

- `uc()` — делает все символы строки прописными буквами:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";

$str = "очень длинная строка";
print uc($str); # ОЧЕНЬ ДЛИННАЯ СТРОКА
```

- `lc()` — делает все символы строки строчными буквами:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";
```

```
$str = "ОЧЕНЬ ДЛИННАЯ СТРОКА";
print lc($str); # очень длинная строка
```

- `ucfirst()` — делает первый символ строки прописным:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";

$str = "очень длинная строка";
print ucfirst($str); # Очень длинная строка
```

- `lcfirst()` — делает первый символ строки строчным:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";

$str = "ОЧЕНЬ ДЛИННАЯ СТРОКА";
print lcfirst($str); # оЧЕНЬ ДЛИННАЯ СТРОКА
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Функции `lc()`, `uc()`, `lcfirst()` и `ucfirst()` правильно обрабатывают буквы русского алфавита только после настройки локали.

- `split()` — разделяет строку на подстроки по указанному разделителю и добавляет их в массив. Функция позволяет использовать регулярные выражения. Имеет следующий формат:

```
split(/<Разделитель>/, <Строка>, <Лимит>);

$str = "Фамилия\tИмя\tОтчество\tГод рождения";
@Mass = split(/\t/, $str);
$str2 = join(", ", @Mass);
```

```
print "$str2";
# Выведет Фамилия, Имя, Отчество, Год рождения
```

Можно указать сразу несколько разделителей в квадратных скобках:

```
$str = "Фамилия\tИмя\tОтчество\nГод рождения";
@Mass = split(/[\t\n ]/, $str);
$str2 = join(", ", @Mass);
print "$str2";
# Выведет Фамилия, Имя, Отчество, Год, рождения
```

2.5.3. Настройка локали

При изменении регистра русских букв может возникнуть проблема. Чтобы ее избежать, необходимо правильно настроить локаль (совокупность локальных настроек системы).

Для установки локали используется функция `Setlocale()`. Функция имеет следующий формат:

```
setlocale(<Категория>, <Локаль>);
```

Параметр `<Категория>` может принимать следующие значения:

- `LC_ALL` — устанавливает локаль для всех режимов;
- `LC_COLLATE` — для сравнения строк;
- `LC_CTYPE` — для перевода символов в нижний или верхний регистр;
- `LC_MONETARY` — для отображения денежных единиц;
- `LC_NUMERIC` — для форматирования дробных чисел;
- `LC_TIME` — для форматирования вывода даты и времени.

Примеры:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# Настройка локали
use locale;
use POSIX 'locale_h';
setlocale(LC_CTYPE, 'ru_RU.CP1251');
# Конец настройки локали
print "Content-type: text/html\n\n";

$str = "очень длинная строка";
print uc($str); # ОЧЕНЬ ДЛИННАЯ СТРОКА
```

2.5.4. Функции для работы с символами

Рассмотрим функции `chr()` и `ord()`:

- `chr(<Код символа>)` — возвращает символ по указанному коду:

```
print chr(81); # Выведет Q
```

- `ord(<Символ>)` — возвращает код указанного символа:

```
print ord("Q"); # Выведет 81
```

2.5.5. Поиск в строке

Рассмотрим функции `index()` и `rindex()`:

- `index()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстроки нет в строке, то функция возвращает значение –1. Функция зависит от регистра символов. Имеет следующий формат:

```
index(<Строка>, <Подстрока>, [<Начальная позиция поиска>])
```

  Если начальная позиция не указана, то поиск будет производиться с начала строки:

```
if (index("Привет", "При") != -1) {
   print "Найдено";
}
else { print "Не найдено"; }
# Выведет "Найдено"
if (index("Привет", "при") != -1) {
   print "Найдено";
}
else { print "Не найдено"; }
# Выведет "Не найдено"
```

- `rindex()` — возвращает номер позиции последнего вхождения подстроки в строку. Если подстроки нет в строке, то функция возвращает значение –1. Функция зависит от регистра символов. Имеет следующий формат:

```
rindex(<Строка>, <Подстрока>, [<Начальная позиция поиска>])

$str = "строка строка строка";
# Применение функции index()
print index($str, "строка"); # Выведет 0
# Применение функции rindex()
print "<BR>", rindex($str, "строка"); # Выведет 14
```

2.5.6. Оператор трансляции *tr///*

Оператор трансляции `tr///` подставляет одни заданные символы вместо других. Имеет следующий формат:

```
tr/<Символы для замены>/<Новые символы>/[<Модификатор>]
```

В параметре `<Модификатор>` могут быть указаны следующие флаги:

- без флага — символ из параметра `<Символы для замены>` будет заменен на соответствующий по порядку символ из параметра `<Новые символы>`. Если список новых символов короче, чем список символов для замены, то вместо недостающих символов будет использоваться последний символ из списка новых символов;
- `c` — будут заменены все символы, кроме указанных в параметре `<Символы для замены>`;
- `d` — удаляются все символы, для которых нет соответствия в параметре `<Новые символы>`;
- `s` — последовательности одинаковых символов, которые образовались в результате трансляции, будут заменены одним символом.

Для преобразования строки используется оператор связывания `=~`.

Преобразуем все строчные буквы в прописные:

```
$str = "строка";
$str =~ tr/а-я/А-Я/;
print $str; # Выведет СТРОКА
```

Заменим все символы, не являющиеся русскими буквами, на знак вопроса:

```
$str = "строка string";
$str =~ tr/а-я /?/c;
print $str; # Выведет "строка ??????"
```

Вместо шести знаков вопроса выведем только один:

```
$str = "строка string";
$str =~ tr/а-я /?/cs;
print $str; # Выведет "строка ?"
```

А теперь выведем количество произведенных замен:

```
$str = "строка";
$count = $str =~ tr/а-я/А-Я/;
print $count; # Выведет 6
```

Сделаем список для замены больше на один символ, чем список новых символов:

```
$str = "строка";
$str =~ tr/строка/строк/;
print $str; # Выведет строкк
```

Как видно из примера, букве "а" не нашлось соответствующего символа для замены. По этой причине этот символ был заменен последней буквой из списка новых символов ("к"). Теперь продемонстрируем результат применения модификатора "d":

```
$str = "строка";
$str =~ tr/строка/строк/d;
print $str; # Выведет строк
```

В этом случае буква "а" была просто удалена.

Если список новых символов не указан, то символы не заменяются Точнее, они заменяются сами на себя, но это не заметно при выводе. Именно благодаря замене можно посчитать количество вхождений символа в строку:

```
$str = "ссылка";
$count = $str =~ tr/с//;
print $count; # Выведет 2
print $str; # Выведет ссылка
```

2.5.7. Кодирование строки

Для кодирования строки применяется модуль `Digest::MD5`. Алгоритм MD5 используется для кодирования паролей. Для шифрования используются три функции: `md5()`, `md5_hex()` и `md5_base64()`. Функция `md5()` возвращает строку из 16 символов, `md5_hex()` — из 32 символов, а `md5_base64` — из 22. Если необходима совместимость с PHP-функцией `md5()`, то следует шифровать пароль с помощью функции `md5_hex()`. Приведем вывод каждой функции:

```
use Digest::MD5 qw(md5 md5_hex md5_base64);
$pass = "password";
$pass1 = md5($pass); # _ММ;Z§eЦ□ѓ'Юё,П™
$pass2 = md5_hex($pass); # 5f4dcc3b5aa765d61d8327deb882cf99
$pass3 = md5_base64($pass); # X03MO1qnZdYdgyfeuILPmQ
```

Для проверки введенного пользователем пароля необходимо зашифровать введенный пароль, а затем произвести сравнение с сохраненным в базе (листинг 2.24).

Листинг 2.24. Проверка правильности пароля

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

```
use CGI::Carp qw(fatalsToBrowser);
# Шифрование Md5
use Digest::MD5 qw( md5_hex );
print "Content-type: text/html\n\n";

$pass = "5f4dcc3b5aa765d61d8327deb882cf99"; # Пароль, сохраненный в базе
$pass2 = "password"; # Пароль, введенный пользователем
if ($pass eq md5_hex($pass2)) { # Проверка правильности пароля
   print "Пароль правильный";
}
```

Для шифрования можно воспользоваться объектным стилем. Обратите внимание, результат отличается от результата, возвращаемого функциями.

```
use Digest::MD5;
$pass = "password";
$dg = Digest::MD5->new;
$dg->add($pass);
$pass1 = $dg->digest; # _ММ;Z$eЦ□ѓ'Юё,П™
$pass2 = $dg->hexdigest; # d41d8cd98f00b204e9800998ecf8427e
$pass3 = $dg->b64digest; # 1B2M2Y8AsgTpgAmY7PhCfg
```

2.5.8. Регулярные выражения

Регулярные выражения позволяют осуществить сложный поиск. Использовать регулярные выражения позволяют операторы `m//` и `s///`.

Оператор `m//` выполняет поиск в строке с помощью регулярного выражения. Имеет следующий формат:

```
m/<Регулярное выражение>/[<Модификатор>]
```

В параметре `<Модификатор>` могут быть указаны следующие флаги:

- `i` — поиск без учета регистра;
- `m` — поиск в строке, состоящей из нескольких строк, разделенных символом новой строки;
- `s` — однострочный режим. В этом режиме символ точки сопоставляется с символом новой строки;
- `x` — разрешает использовать в регулярном выражении пробелы и комментарии;
- `g` — поиск всех соответствий шаблону;
- `c` — не переустанавливать позицию поиска после неудачного сопоставления. Используется только в комбинации с флагом `g`.

Можно использовать несколько вариантов символов разделителей:

```
m#<Регулярное выражение>#[<Модификатор>]
m(<Регулярное выражение>)[<Модификатор>]
m{<Регулярное выражение>}[<Модификатор>]
m[<Регулярное выражение>][<Модификатор>]
/<Регулярное выражение>/[<Модификатор>]
```

Обратите внимание на последний вариант. Если используется символ `/`, то операция поиска подразумевается по умолчанию.

Для поиска применяются два оператора связывания:

- `=~` — если соответствие найдено, то оператор возвратит `true`;
- `!~` — если соответствие найдено, то оператор возвратит `false`.

Если перед выражением указан массив, то в нем сохраняются соответствия подвыражений с шаблоном:

```
$str = "<BR><TD><OL>";
$pattern = qr/<(\w+)>/;
@Mass = $str =~ m/$pattern/i;
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] <BR>";
}
```

В данном примере выведется только первое соответствие с шаблоном, т. к. не указан флаг глобального поиска `g`. Классу `\w` соответствуют любые буквы и цифры, а знак `+` указывает, что искомый символ должен встретиться хотя бы один раз (возможно много раз). В массив попадают только те значения, которые заключены в круглые скобки. Таким образом, мы можем выбрать из строки названия всех тегов:

```
$str = "<BR><TD><OL>";
$pattern = qr/<(\w+)>/;
@Mass = $str =~ m/$pattern/ig;
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] <BR>";
}
```

Вывод:

```
BR
TD
OL
```

В этих примерах мы использовали оператор `qr//`. С его помощью переменная `$pattern` будет содержать откомпилированное регулярное выражение. Оператор имеет следующий формат:

```
qr/<Регулярное выражение>/[<Модификатор>]
```

В параметре <Модификатор> могут быть указаны следующие флаги: i, m, s, x. Их значения такие же, как и у флагов оператора поиска m//.

При использовании глобального поиска функция pos() возвращает текущую позицию, с которой будет начинаться сопоставление:

```
$str = "<BR><TD><OL>";
$pattern = qr/<(\w+)>/;
while ($str =~ m/$pattern/ig) {
   print "$1 — позиция ", pos($str), "<BR>";
}
```

Вывод:

```
BR — позиция 4
TD — позиция 8
OL — позиция 12
```

В этом коде мы использовали специальную переменную $1, которая содержит фрагмент (соответствующий шаблону), заключенный в регулярном выражении в круглые скобки. После знака $ указывается число, соответствующее порядковому номеру круглых скобок в строке (значение во вторых круглых скобках будет сохранено в переменной $2, в третьих — в $3 и т. д.). Для примера произведем проверку правильности ввода E-mail и выведем отдельно название почтового ящика и название домена:

```
$emails = 'unicross@mail.ru';
$pattern = qr/^([a-z0-9_\.\-]+)\@(([a-z0-9\-]+\.)+[a-z]{2,4})$/;
if ($emails =~ m/$pattern/i) {
   print "E-mail соответствует шаблону";
   print "<BR>ящик: ", $1, " домен: ", $2;
}
else {
   print "Не соответствует шаблону";
}
```

Вывод:

```
E-mail соответствует шаблону
ящик: unicross домен: mail.ru
```

Обратите внимание: мы заключили строку с E-mail в апострофы, т. к. символ @ является специальным. Если заключить в кавычки, то все, что расположено после символа @, будет считаться названием массива и соответственно будет заменено его значением.

Оператор s/// выполняет поиск и замену в исходной строке с помощью регулярного выражения. Имеет следующий формат:

```
s/<Регулярное выражение>/<Строка для замены>/[<Модификатор>]
```

В параметре <Модификатор> могут быть указаны следующие флаги:

- i — поиск без учета регистра;
- m — поиск в строке, состоящей из нескольких строк, разделенных символом новой строки;
- s — однострочный режим. В этом режиме символ точка сопоставляется с символом новой строки;
- x — разрешает использовать в регулярном выражении пробелы и комментарии;
- g — поиск и замена всех соответствий шаблону;
- e — указывает, что в параметре <Строка для замены> указано выражение языка Perl, которое необходимо предварительно вычислить.

Так же, как и в операции поиска, все фрагменты, заключенные в круглые скобки, попадают в специальные переменные $n, где n — порядковый номер скобок в шаблоне. Для примера возьмем два тега и поменяем имена тегов местами:

```
$str = "<BR><TD>";
$str =~ s/<(\w+)><(\w+)>/<$2><$1>/i;
print $str;
```

Вывод в исходном HTML-коде:

```
<TD><BR>
```

Такой же результат можно получить, если вместо знака $ указать символ \:

```
$str = "<BR><TD>";
$str =~ s/<(\w+)><(\w+)>/<\2><\1>/i;
print $str;
```

Для примера, заменим все палиндромы из пяти букв в строке на слово "палиндром":

```
$str = "потоп комок слово";
$str =~ s/([а-я])([а-я])[а-я]\2\1/палиндром/i;
print $str; # Выведет палиндром комок слово
```

Как видно из примера, был заменен только один палиндром. Для того чтобы были заменены все палиндромы, необходимо указать флаг глобального поиска g:

```
$str = "потоп комок слово";
$str =~ s/([а-я])([а-я])[а-я]\2\1/палиндром/ig;
print $str; # Выведет палиндром палиндром слово
```

Функция `split()` также поддерживает регулярные выражения. Она разделяет строку на подстроки по указанному разделителю и добавляет их в массив:

```
$str = 'unicross@mail.ru';
@Mass = split("[\@\.]", $str);
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] <BR>";
}
```

Вывод:

```
unicross
mail
ru
```

Метасимволы, используемые в регулярных выражениях:

- `^` — привязка к началу строки;
- `$` — привязка к концу строки:

```
$str = "2";
if ($str =~ /^[0-9]+$/) {
   print "Число"; # Выведет "Число"
}
else {
   print "Не число";
}
$str = "Строка2";
if ($str =~ /^[0-9]+$/) {
   print "Число";
}
else {
   print "Не число"; # Выведет "Не число"
}
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая число, вернет "Число":

```
$str = "Строка2";
if ($str =~ /[0-9]+/) {
   print "Число"; # Выведет "Число"
}
else {
   print "Не число";
}
```

Можно указать привязку только к началу или только к концу строки:

```
$str = "Строка2";
if ($str =~ /[0-9]+$/) {
   print "Есть число в конце строки";
}
else {
   print "Нет числа в конце строки";
}
# Выведет "Есть число в конце строки"
if ($str =~ /^[0-9]+/) {
   print "Есть число в начале строки";
}
else {
   print "Нет числа в начале строки";
}
# Выведет "Нет числа в начале строки"
```

- `[]` — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире:
  - `[09]` — соответствует числу 0 или 9;
  - `[0-9]` — соответствует любому числу от 0 до 9;
  - `[абв]` — соответствует буквам "а", "б" и "в";
  - `[а-г]` — соответствует буквам "а", "б", "в" и "г";
  - `[а-я]` — соответствует любой букве от "а" до "я";
  - `[АБС]` — соответствует буквам "А", "Б" и "С";
  - `[А-Я]` — соответствует любой букве от "А" до "Я";
  - `[а-яА-Я]` — соответствует любой букве в любом регистре;
  - `[0-9а-яА-Яa-zA-Z]` — любая цифра и любая буква независимо от регистра и языка.

  Значение можно инвертировать, если после первой скобки указать символ `^`. Таким образом можно указать символы, которых не должно быть на этом месте в строке:
  - `[^09]` — не число 0 или 9;
  - `[^0-9]` — не число от 0 до 9;
  - `[^а-яА-Яa-zA-Z]` — не буква.

Вместо перечисления символов можно использовать стандартные классы:

- `\d` — соответствует любой цифре;
- `\w` — соответствует любой латинской букве, цифре и знаку подчеркивания;
- `\s` — любой пробельный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- . ("точка") — любой символ, кроме символа перевода строки (`\n`);
- `\D` — не цифра;
- `\W` — не латинская буква, не цифра и не знак подчеркивания;
- `\S` — не пробельный символ.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Метасимвол `\w` работает только с буквами латинского алфавита. С буквами русского языка он не употребляется.

Что же делать, если нужно найти точку, ведь символ точки соответствует любому символу, кроме символа перевода строки? Для этого перед специальным символом необходимо указать обратный слэш `\`. Продемонстрируем это на примере (листинг 2.25).

Листинг 2.25. Проверка правильности введенной даты

```
$str = "29,04.2007"; # Неправильная дата (вместо точки указана запятая)

$pattern = qr/^[0-3]\d.[01]\d.[12][09]\d\d$/;
# Символ "\" не указан перед точкой
if ($str =~ /$pattern/) { print "Дата введена правильно"; }
else { print "Дата введена неправильно"; }
# Выведет "Дата введена правильно", точка означает любой символ

$pattern = qr/^[0-3]\d\.[01]\d\.[12][09]\d\d$/;
# Символ "\" указан перед точкой
if ($str =~ /$pattern/) { print "Дата введена правильно"; }
else { print "Дата введена неправильно"; }
# Выведет "Дата введена неправильно",
# перед точкой указан символ "\"
```

Количество вхождений символа в строку задается с помощью *квантификаторов*:

- `{n}` — соответствует `n` вхождениям символа в строку, например, `\d{2}` соответствует двум вхождениям любой цифры;

- `{n,}` — по крайней мере, `n` вхождений символа в строку, например, `\d{2,}` соответствует двум и более вхождениям любой цифры;
- `{n,m}` — не менее `n` вхождений символа в строку и не более `m`. Цифры указываются через запятую без пробела, например, `\d{2,5}` означает от двух до пяти вхождений любой цифры;
- `*` — возможность вхождения символа, например, `\d*` — цифры могут не встретиться в строке или встретиться много раз;
- `+` — ненулевое количество вхождений символа в строку. `\d+` — цифра может встретиться один раз или встретиться много раз;
- `?` — ноль или одно число вхождений символа в строку, например, `\d?` — цифра может встретиться один раз или не встретиться совсем.

Регулярное выражение можно разбить на подвыражения с помощью круглых скобок. Каждая группа символов, соответствующих подвыражению, сохраняется в памяти. В дальнейшем группу символов можно извлечь с помощью следующего синтаксиса:

```
\{Номер группы}
```

Нумерация групп символов осуществляется согласно их появлению в регулярном выражении:

```
$pattern = qr/<(.+)>(.*)<\/\1>/;
$str = "<B><U>Подчеркнутый полужирный текст</U></B>";
@Mass = $str =~ /$pattern/;
for ($i=0; $i<@Mass; $i++) {
   print $Mass[$i], "\n";
}
```

- <.+> — соответствует любому открывающему тегу;
- <(.+)> — с помощью скобок запоминаем имя тега;
- <\/\1> — ищем соответствующий закрывающий тег, который был найден в первых скобках;
- (.*) — сохраняем группу символов между открывающим и закрывающим тегами.

```
$pattern = qr/<(.+)>(.*)<\/\1>/;
$str = "<B><U>Подчеркнутый полужирный текст</U></B>";
@Mass = $str =~ /$pattern/;
for ($i=0; $i<@Mass; $i++) {
   print $Mass[$i], "\n";
}
```

В исходном HTML-коде отобразится:

```
B
<U>Подчеркнутый полужирный текст</U>
```

Первая строка — это символ в первых скобках (B), вторая строка — группа символов во вторых скобках ("<U>Подчеркнутый полужирный текст</U>").

С помощью круглых скобок можно объединять метасимволы в группы. Рассмотрим это на примере проверки правильности ввода E-mail-адреса (листинг 2.26).

Листинг 2.26. Проверка корректности адреса электронной почты

```
$emails = 'unicross@mail.ru';
$pattern = qr/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/i;
if ($emails =~ m/$pattern/) {
   print "E-mail правильный";
}
else {
   print "E-mail не правильный";
}
```

Итак, этому шаблону соответствует любой E-mail:

```
/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/i
```

Сравнение производится без учета регистра. Метасимвол ^ указывает привязку к началу строки, а $ — привязку к концу строки. E-mail разбивается на три части:

- `[a-z0-9_\.\-]+` — имя ящика, указанное до символа `@`;
- `([a-z0-9\-]+\.)+` — имя поддомена, указанное после символа `@`, но до названия зоны. Так как поддоменов может быть много, подвыражение `[a-z0-9\-]+\.` заключается в круглые скобки, после которых указывается метасимвол `+`, указывающий, что подвыражение может встречаться один и более раз;
- `[a-z]{2,4}` — название зоны может содержать только от 2 до 4 букв (ru, com, info).

> ***Обратите внимание***
>
> Символ `@` является специальным. По этой причине перед ним необходимо указать защитный слэш.

Рассмотрим еще один пример. Предположим, необходимо получить все значения между одинаковыми тегами:

```
$pattern = qr/<S1>(.*)<\/S1>/;
$str = "<S1>Значение1</S1><B>Лишнее значение</B><S1>Значение2</S1>";
@Mass = $str =~ /$pattern/g;
for ($i=0; $i<@Mass; $i++) {
   print $Mass[$i], "<BR>";
}
```

Вместо желаемого результата мы получим:

```
Значение1</S1><B>Лишнее значение</B><S1>Значение2
```

Такое поведение квантификаторов называется "жадностью". Чтобы ограничить эту "жадность", необходимо после символа `*` указать символ `?`:

```
$pattern = qr/<S1>(.*?)<\/S1>/;
$str = "<S1>Значение1</S1><B>Лишнее значение</B><S1>Значение2</S1>";
@Mass = $str =~ /$pattern/g;
for ($i=0; $i<@Mass; $i++) {
   print $Mass[$i], "<BR>";
}
```

В этом случае мы получим только все значения между нужными тегами:

```
Значение1
Значение2
```

С помощью символа `?` можно ограничить "жадность" и других квантификаторов.

Логическое ИЛИ рассмотрим на примере выражения `n|m` (соответствует одному из символов `n` или `m`):

`красн(ая)|(ое)` — `красная` **или** `красное`, **но не** `красный`.

2.5.9. Выполнение команд, содержащихся в строке

С помощью функции `eval()` можно выполнить строку как Perl-код:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

my ($str, $str1, $str2, $str3, $str4, $str5);
$str1=$str2=$str3=$str4=$str5="Привет";
for (my $i=1; $i<6; $i++) {
```

```
    $str = '$str' . $i . ' = "Строка' . $i . '";';
    eval($str);
}
print "$str1<BR>";
print "$str2<BR>";
print "$str3<BR>";
print "$str4<BR>";
print "$str5<BR>";
```

Вывод:

```
Строка1
Строка2
Строка3
Строка4
Строка5
```

Если в строке содержатся ошибки, то их описание сохраняется в специальной переменной `$@`. Если ошибок не возникло, то переменная содержит пустую строку. В качестве примера создадим код с ошибкой и выведем описание ошибки:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

my $str = 'prinnt "Hello, world";';
eval($str);
print $@ if (defined($@));

$str = 'print "<BR>Hello, world";';
eval($str);
print $@ if (defined($@));
```

Вывод:

```
syntax error at (eval 2) line 1, near "prinnt "Hello, world""
Hello, world
```

Как видно из примера, синтаксическая ошибка не привела к остановке выполнения всей программы. Благодаря этому многие программисты используют функцию `eval()` для обработки исключительных ситуаций в своих программах.

Создадим программу, позволяющую выполнять любые команды, вводимые в поле TEXTAREA (листинг 2.27).

Листинг 2.27. Выполнение команд, содержащихся в строке

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

my $cod = param("cod");
print "<HTML><HEAD>\n";
print "<TITLE>Выполнение команд, содержащихся в строке</TITLE>\n";
print "</HEAD><BODY>\n";
if (defined($cod)) {
   print "<B>Результат выполнения команды</B><BR><BR>\n";
   eval($cod);
   print $@ if (defined($@));
   print "<BR><BR>\n";
}
print "<B>Введите команду для выполнения</B><BR><BR>\n";
print "<FORM>\n";
print "<TEXTAREA name=\"cod\" cols=\"25\" rows=\"10\">";
print "</TEXTAREA><BR><BR>\n";
print "<INPUT type=\"submit\" value=\"Выполнить код\">\n";
print "</FORM>\n";
print "</BODY></HTML>\n";
```

Внимание!

Никогда не подставляйте данные, вводимые пользователями, в функцию eval(), т. к. это потенциальная угроза безопасности. Следует помнить, что некоторые команды могут полностью уничтожить ваш Web-сайт.

2.6. Функции для работы с числами

Перечислим основные функции для работы с числами:

- sin(), cos() — стандартные тригонометрические функции (синус, косинус);
- exp() — экспонента;

- `log()` — натуральный логарифм;
- `sqrt()` — квадратный корень;
- `abs()` — абсолютное значение;
- `int()` — возвращает целую часть числа;
- `hex()` — переводит шестнадцатеричное число в десятичное;
- `oct()` — переводит восьмеричное число в десятичное;
- `rand([<Конец диапазона>])` — возвращает случайное число от 0 до `<Конец диапазона>`. Если параметр не задан, то его значение равно 1;

```
print int(rand(100));
```

- `srand()` — настраивает генератор случайных чисел на новую последовательность:

```
srand();
print int(rand(100));
```

Создадим генератор паролей произвольной длины (листинг 2.28). Для этого в массив `@mass` добавляем все разрешенные символы, а далее в цикле получаем содержимое массива по случайному индексу. По умолчанию будет выдаваться пароль из 8 символов.

Листинг 2.28. Генератор паролей

```
sub f_passw_generator {
   $count_char = shift();
   $count_char=8 if (!defined($count_char));
   @mass = ('a','b','c','d','e','f','g','h','i','j','k','l','m',
   'n','o','p','q','r','s','t','u','v','w','x','y','z',
   'A','B','C','D','E','F','G','H','I','J','K','L',
   'M','N','O','P','Q','R','S','T','U','V', 'W',
   'X','Y','Z','1','2','3','4','5','6','7','8','9','0');
    $passw = "";
    for($i=0;$i<$count_char;$i++) {
      $passw .= $mass[int(rand(scalar(@mass)))];
    }
return $passw;
}
print f_passw_generator(10); # Выведет примерно GLa1OpdkTr
```

Рассмотрим каждую строку программы. В первой строке мы определяем функцию `f_passw_generator()` с помощью ключевого слова `sub`. Далее получаем значение количества символов с помощью следующей строки:

```
$count_char = shift();
```

Все параметры, переданные функции, попадают в специальный массив `@_`. С помощью функции `shift()` мы присваиваем первый элемент массива переменной `$count_char`. Обратите внимание, в функции `shift()` не указан параметр. По умолчанию этим параметром является массив `@_`. Иными словами, предыдущий фрагмент кода можно переписать так:

```
$count_char = shift(@_);
```

Далее мы проверяем, определена ли переменная `$count_char` с помощью функции `defined()`. Если переменная определена, функция возвращает `true`, мы инвертируем возвращаемое значение при помощи выражения:

```
!defined($count_char)
```

Если переменная не определена, то ей присваивается значение 8. Таким образом, если мы не укажем параметр при вызове функции `f_passw_generator()`, то длина пароля по умолчанию будет равна восьми символам. В следующих строках мы заполняем массив `@mass` разрешенными символами. Далее присваиваем переменной `$passw` пустую строку. Затем в цикле определяем случайный индекс массива и по нему получаем значение элемента массива. С помощью выражения `scalar(@mass)` мы получаем количество элементов массива. Так как функция `rand()` возвращает вещественное число, большее или равное 0 и меньшее числа, заданного параметром, мы с помощью функции `int()` возвращаем целую часть числа:

```
int(rand(scalar(@mass)))
```

Например, если число элементов массива равно 10, то мы получим случайное число от 0 до 9. Учитывая, что индексация массива начинается с 0, то округление числа элементов массива даст максимальный индекс массива. По этому индексу определяем значение элемента массива:

```
$mass[int(rand(scalar(@mass)))]
```

Полученный символ добавляем к значению переменной `$passw` при помощи операции конкатенации строк (`.=`). Далее с помощью ключевого слова `return` мы возвращаем полученный пароль в место вызова функции:

```
return $passw;
```

Для вызова функции указываем ее название, а в круглых скобках задаем количество символов в пароле. В итоге сгенерированный пароль выводим в окно Web-браузера с помощью оператора `print`:

```
print f_passw_generator(10);
```

2.7. Функции для работы с датой и временем

Функция `time()` возвращает количество секунд, прошедшее с 1 января 1970 г.:

```
print time(); # Выведет примерно 1211129478
```

Функция `gmtime()` возвращает массив значений даты и времени, аргументом является количество секунд, полученное с помощью функции `time()`. Соответствует временной зоне Гринвича:

```
@mass = gmtime(time());
for($i=0;$i<@mass;$i++) {
   print $mass[$i], "<BR>";
}
```

Функция `localtime()` возвращает массив значений даты и времени, аргументом является количество секунд, полученное с помощью функции `time()`. Соответствует зоне местного времени:

```
@mass = localtime(time());
for($i=0;$i<@mass;$i++) {
   print $mass[$i], "<BR>";
}
```

Функции `gmtime()` и `localtime()` возвращают массив следующих значений:

- `0` — секунды (от 0 до 59);
- `1` — минуты (от 0 до 59);
- `2` — часы (от 0 до 23);
- `3` — число месяца (от 1 до 31);
- `4` — месяц (0 — для января, 11 — для декабря);
- `5` — год. К полученному значению необходимо прибавить 1900;
- `6` — день недели (0 — для воскресенья, 6 — для субботы);
- `7` — порядковый номер дня в году (от 0 до 365).

Выведем текущую дату и время таким образом, чтобы день недели и месяц были написаны по-русски (листинг 2.29).

Листинг 2.29. Вывод текущей даты

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

```
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

@day = ("воскресенье", "понедельник", "вторник", "среда", "четверг",
"пятница", "суббота");
@month = ("января", "февраля", "марта", "апреля", "мая", "июня",
"июля", "августа", "сентября", "октября", "ноября", "декабря");
@date = localtime(time());
$year = $date[5] + 1900;
$date = "Сегодня<BR>";
$date .= $day[$date[6]];
$date .= " " . $date[3] . " ";
$date .= $month[$date[4]];
$date .= " " . $year . " ";
if (length($date[1]) == 1) {
   $date[1] = "0" . $date[1];
}
if (length($date[0]) == 1) {
   $date[0] = "0" . $date[0];
}
$date .= $date[2] . ":" . $date[1] . ":" . $date[0];
print $date;
```

Вывод:

```
Сегодня
воскресенье 18 мая 2008 20:53:54
```

Если в функциях `gmtime()` и `localtime()` указан параметр, то дата будет не текущая, а соответствующая значению этого параметра. Например, в функции можно передать количество секунд, прошедших с 1 января 1970 г., и получить любое другое форматирование даты:

```
@date = localtime(1189210561);
```

Для форматирования даты и времени можно воспользоваться функцией `strftime()` из модуля `POSIX`.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Вывод функции `strftime()` зависит от настройки локали.

Функция имеет следующий формат:

```
strftime(<Строка формата даты/времени>, <Массив значений даты>)
```

В качестве параметра `<Массив значений даты>` обычно указывается функция `gmtime()` или функция `localtime()`. В параметре `<Строка формата даты/времени>` могут быть использованы следующие модификаторы:

- `%c` — представление даты и времени для текущей локали (например, 18.06.2008 23:14:48);
- `%x` — представление даты для текущей локали (например, 18.06.2008);
- `%X` — представление времени для текущей локали (например, 23:18:28);
- `%Y` — год из 4 цифр;
- `%y` — год из 2 цифр;
- `%j` — день с начала года (от "001" до "366");
- `%b` — сокращенное название месяца (например, "июн" для июня);
- `%B` — полное название месяца (например, июнь);
- `%m` — номер месяца с предваряющим нулем (от "01" до "12");
- `%d` — номер дня с предваряющим нулем (от "01" до "31");
- `%a` — сокращенное название дня недели (например, "Ср" для среды);
- `%A` — полное название дня недели (например, среда);
- `%H` — часы в 24-часовом формате (от "00" до "23");
- `%I` — часы в 12-часовом формате (от "01" до "12");
- `%M` — минуты (от "00" до "59");
- `%S` — секунды (от "00" до "59").

Выведем текущую дату и время с помощью функции `strftime()` для локали `ru_RU.CP1251`:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use POSIX;
# Настройка локали
use locale;
setlocale(LC_ALL, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";

print strftime("Сегодня %A %d.%m.%Y %H:%M:%S", localtime());
```

Вывод:

```
Сегодня среда 18.06.2008 23:14:48
```

2.8. Условные операторы

Условные операторы позволяют, в зависимости от значения логического выражения, выполнить отдельный участок программы или, наоборот, не выполнять его. Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь).

2.8.1. Операторы сравнения

Операторы сравнения используются в логических выражениях.

> ***Обратите внимание***
> Для чисел и строк операторы сравнения являются разными.

Операторы сравнения для чисел:

- `==` — равно;
- `!=` — не равно;
- `<` — меньше;
- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно;
- `<=>` — возвращает следующие значения:
  - `0` — если значения равны;
  - `1` — если левое значение больше правого;
  - `-1` — если левое значение меньше правого.

Операторы сравнения для строк:

- `eq` — равно;
- `ne` — не равно;
- `lt` — меньше;
- `gt` — больше;
- `le` — меньше или равно;
- `ge` — больше или равно;
- `cmp` — возвращает следующие значения:
  - `0` — если строки равны;
  - `1` — если левая строка больше правой;
  - `-1` — если левая строка меньше правой.

Значение логического выражения можно инвертировать с помощью оператора !:

```
!($var1 == $var2)
```

Если переменные $var1 и $var2 равны, то возвращается значение true, но т. к. перед выражением стоит оператор !, выражение вернет false.

Можно несколько логических выражений объединить в одно большое с помощью следующих операторов:

- && — логическое И;
- || — логическое ИЛИ.

```
($var1 == $var2) && ($var2 != $var3)
($var1 == $var2) || ($var3 == $var4)
```

В первом случае выражение вернет true только в случае, если оба выражения вернут true. Второе выражение вернет true, если хотя бы одно из выражений вернет true.

Вместо оператора && можно использовать логическую операцию AND, а вместо || — логическую операцию OR:

- AND — логическое И;
- OR — логическое ИЛИ.

```
($var1 == $var2) AND ($var2 != $var3)
($var1 == $var2) OR ($var3 == $var4)
```

2.8.2. Оператор ветвления *if...else*

Оператор ветвления мы уже использовали ранее в наших примерах, например, чтобы узнать, определена ли переменная. Если переменная определена, функция defined() возвращает значение true, это условие можно проверить, используя оператор ветвления if...else:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$name = param("name");
print "<HTML><HEAD>\n";
print "<TITLE>Первая программа</TITLE>\n";
```

```
print "</HEAD><BODY>\n";
if (defined($name)) {
   print "Hello, $name";
}
else {
   print "Введите ваше имя<BR>\n";
   print "<FORM>\n";
   print "<INPUT type=\"text\" name=\"name\">\n";
   print "<INPUT type=\"submit\" value=\"OK\">\n";
   print "</FORM>\n";
}
print "</BODY></HTML>\n";
```

Обратите внимание, что логическое выражение не содержит операторов сравнения:

```
if (defined($name)) {
```

Проверка на равенство выражения значению `true` (Истина) выполняется по умолчанию.

Оператор ветвления `if...else` имеет следующий формат:

```
if (<Логическое выражение>) {
<Блок, выполняемый, если условие истинно>
}
[elsif (<Логическое выражение>) {
<Блок, выполняемый, если условие истинно>
}]
[else {
<Блок, выполняемый, если все условия ложны>
}]
```

Если блок состоит из одного выражения, то можно воспользоваться следующим форматом:

```
<Блок, выполняемый, если условие истинно> if (<Логическое выражение>);
```

Для примера: напишем программу, которая в зависимости от введенного пользователем числа проверяет, является ли число четным или нет. После проверки выводится соответствующее сообщение (листинг 2.30).

Листинг 2.30. Проверка числа на четность

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
```

```
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$var = param("var");
print "<HTML><HEAD>\n";
print "<TITLE>Проверка числа на четность</TITLE>\n";
print "</HEAD><BODY>\n";
print "<B>Проверка числа на четность</B><BR><BR>\n";
# Выводим форму
print "Введите число<BR>\n";
print "<FORM>\n";
print "<INPUT type=\"text\" name=\"var\">\n";
print "<INPUT type=\"submit\" value=\"OK\">\n";
print "</FORM><BR>\n";
if (defined($var)) {
   if ($var =~ /^[0-9]+$/) {
      if ($var%2 == 0) {
         print "$var — Четное число";
      }
      else {
         print "$var — Нечетное число";
      }
   }
   else { print "Необходимо ввести число"; }
}
print "</BODY></HTML>\n";
```

Как видно из примера, один условный оператор можно вложить в другой. Более того, блока `else` может не быть совсем:

```
print "$var — Четное число" if ($var%2 == 0);
```

Кроме того, оператор `if...else` позволяет проверить сразу несколько условий. Рассмотрим это на примере (листинг 2.31).

Листинг 2.31. Проверка выбранного элемента из списка

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";
```

```
$os = param("os");
print "<HTML><HEAD>\n";
print "<TITLE>Проверка выбранного элемента из списка</TITLE>\n";
print "</HEAD><BODY>\n";
print "<B>Какой операционной системой вы пользуетесь?</B><BR><BR>\n";
# Выводим форму
print "<FORM>\n";
print "<SELECT name=\"os\">\n";
print "<OPTION value=\"0\" selected>Не выбрано\n";
print "<OPTION value=\"1\">Windows 98\n";
print "<OPTION value=\"2\">Windows ME\n";
print "<OPTION value=\"3\">Windows XP\n";
print "<OPTION value=\"4\">Другая\n";
print "</SELECT>\n";
print "<INPUT type=\"submit\" value=\"Выбрал\">\n";
print "</FORM>\n";
if (defined($os)) {
   if ($os=="1") { print "Вы выбрали — Windows 98"; }
   elsif ($os=="2") { print "Вы выбрали — Windows ME"; }
   elsif ($os=="3") { print "Вы выбрали — Windows XP"; }
   elsif ($os=="4") { print "Вы выбрали — Другая"; }
   elsif ($os=="0") { print "Вы не выбрали операционную систему"; }
   else {
      print "Мы не смогли определить вашу операционную систему";
   }
}
print "</BODY></HTML>\n";
```

С помощью оператора `elsif` мы можем определить выбранное значение в списке и вывести соответствующее сообщение.

2.8.3. Оператор *unless*

Оператор `unless` имеет следующий формат:

```
unless (<Логическое выражение>) {
<Блок, выполняемый, если условие ложно>
}
[else {
<Блок, выполняемый если все условие истинно>
}]
```

Если блок состоит из одного выражения, то можно воспользоваться следующим форматом:

```
<Блок, выполняемый, если условие ложно> unless (<Логическое выражение>);
```

Такой же результат можно получить, если указать знак ! перед логическим выражением в операторе if...else:

```
if (!<Логическое выражение>) {
<Блок, выполняемый, если условие истинно>
}
```

В листинге 2.32 приведен пример проверки введенного значения с помощью оператора unless.

Листинг 2.32. Проверка введенного значения

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$var = param("var");
print "<HTML><HEAD>\n";
print "<TITLE>Проверка введенного значения</TITLE>\n";
print "</HEAD><BODY>\n";
# Выводим форму
print "<FORM>\n";
print "<INPUT type=\"text\" name=\"var\">\n";
print "<INPUT type=\"submit\" value=\"OK\">\n";
print "</FORM>\n";
unless ($var =~ /^[0-9]+$/) {
   print "Необходимо ввести число\n";
}
else {
   print "Число введено правильно\n";
}
print "</BODY></HTML>\n";
```

2.8.4. Оператор ?

Оператор ? имеет следующий формат:

```
<Логическое выражение> ? <Выражение если Истина> : <Выражение если Ложь>;
```

Перепишем нашу программу проверки числа на четность и используем оператор ? вместо if...else (листинг 2.33).

Листинг 2.33. Использование оператора ?

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$var = param("var");
print "<HTML><HEAD>\n";
print "<TITLE>Проверка числа на четность</TITLE>\n";
print "</HEAD><BODY>\n";
print "<B>Проверка числа на четность</B><BR><BR>\n";
# Выводим форму
print "Введите число<BR>\n";
print "<FORM>\n";
print "<INPUT type=\"text\" name=\"var\">\n";
print "<INPUT type=\"submit\" value=\"OK\">\n";
print "</FORM><BR>\n";
if (defined($var)) {
   if ($var =~ /^[0-9]+$/) {
      $var%2 == 0 ? print "$var — Четное число" :
      print "$var — Нечетное число";
   }
   else { print "Необходимо ввести число"; }
}
print "</BODY></HTML>\n";
```

2.9. Операторы циклов

Предположим, нужно вывести все числа от 1 до 100 по одному на строке. Обычным способом пришлось бы писать 100 строк кода:

```
print "1<BR>\n";
print "2<BR>\n";
...
print "100<BR>\n";
```

При помощи циклов то же действие можно выполнить одной строкой кода:

```
for ($i=1; $i<101; $i++) { print $i . "<BR>\n"; }
```

Иными словами, циклы позволяют выполнить одни и те же выражения многократно.

2.9.1. Цикл *for*

Цикл `for` используется для выполнения выражений определенное число раз. Имеет следующий формат:

```
for (<Начальное значение>; <Условие>; <Приращение>) {
<Выражения>
}
```

- `<Начальное значение>` — присваивает переменной-счетчику начальное значение;
- `<Условие>` — содержит логическое выражение. Пока логическое выражение возвращает истинное значение, выполняются выражения внутри цикла;
- `<Приращение>` — задает изменение переменной-счетчика при каждой итерации.

Последовательность работы цикла `for`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие, и если оно истинно, выполняются выражения внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращении>`.
4. Происходит переход ко второму шагу.

Цикл выполняется до тех пор, пока `<Условие>` не вернет `false`. Если это не случится, цикл будет бесконечным.

`<Приращение>` может не только увеличивать значение переменной-счетчика, но и уменьшать. Выведем все числа от 100 до 1:

```
for ($i=100; $i>0; $i--) { print $i . "<BR>\n"; }
```

`<Приращение>` может изменять значение переменной-счетчика не только на единицу. Выведем все четные числа от 1 до 100:

```
for ($i=2; $i<101; $i+=2) { print $i . "<BR>\n"; }
```

Для перебора элементов массива можно воспользоваться следующим кодом:

```
@Mass=qw(Один Два Три Четыре);
for($i=0; $i<@Mass; $i++) {
   print "$Mass[$i] ";
} # Один Два Три Четыре
```

Кроме того, можно воспользоваться следующим форматом цикла `for`:

```
for [<Переменная>] (<Массив>) {
<Выражения>
}
```

В этом случае при каждой итерации цикла в переменную будет копироваться один элемент массива:

```
@Mass=qw(Один Два Три Четыре);
for $var (@Mass) {
   print "$var ";
} # Один Два Три Четыре
```

Если переменная не указана, то значение элемента массива сохраняется в специальной переменной `$_`:

```
@Mass=qw(Один Два Три Четыре);
for (@Mass) {
   print "$_ ";
} # Один Два Три Четыре
```

2.9.2. Циклы *while* и *until*

Выполнение выражений в цикле `while` продолжается до тех пор, пока логическое выражение истинно. Имеет следующий формат:

```
<Начальное значение>;
while (<Условие>) {
<Выражения>;
<Приращение>;
}
```

Последовательность работы цикла `while`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие, и если оно истинно, выполняются выражения внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращении>`.
4. Происходит переход ко второму шагу.

Выведем все числа от 1 до 100, используя цикл `while`:

```
$i=1;
while ($i<101) {
   print $i . "<BR>\n";
   $i++;
}
```

Внимание!

Если <Приращение> не указано, то цикл будет бесконечным.

В <Приращении> не обязательно должна быть арифметическая операция. Например, при работе с базами данных в качестве <Приращения> будет перемещение к следующей строке, а условием выхода из цикла — последняя строка в базе данных. В этом случае <Начальным значением> будет первая строка базы данных.

Перебрать элементы ассоциативного массива можно следующим образом:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
while (($key, $value) = each(%Mass)) {
   print "$key => $value ";
} # Выведет Два => 2 Три => 3 Один => 1
```

Цикл until продолжается до тех пор, пока логическое выражение ложно. Выведем все числа от 1 до 100, используя цикл until:

```
$var=1;
$i=1;
until ($var==0) {
   print $i . "<BR>\n";
   $i++;
   $var=0 if ($i>100);
}
```

Цикл until следует применять, когда заранее не известно количество повторений. Выход из цикла происходит, если не выполняется какое-либо условие. В предыдущем примере этим условием является значение переменной $i.

Внимание!

Если условие не указано, то цикл будет бесконечным.

2.9.3. Циклы *do...while* и *do...until*

Выполнение выражений в цикле do...while продолжается до тех пор, пока логическое выражение истинно. Но в отличие от цикла while условие проверяется не в начале цикла, а в конце. По этой причине выражения внутри цикла do...while выполнятся как минимум один раз. Оператор имеет следующий формат:

```
<Начальное значение>;
do {
<Выражения>;
<Приращение>;
} while (<Условие>);
```

Последовательность работы цикла do...while:

1. Переменной-счетчику присваивается начальное значение.
2. Выполняются выражения внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в <Приращении>.
4. Проверяется условие, и если оно истинно, выполняются выражения внутри цикла.
5. Переменная-счетчик изменяется на величину, указанную в <Приращении>.
6. Переход к шагу 4.

Выведем все числа от 1 до 100, используя цикл do...while:

```
$i=1;
do {
   print $i . "<BR>\n";
   $i++;
} while ($i<101);
```

> ***Внимание!***
> Если <Приращение> не указано, то цикл будет бесконечным.

Выполнение выражений в цикле do...until продолжается до тех пор, пока логическое выражение ложно. В отличие от цикла until условие проверяется не в начале цикла, а в конце. Выведем все числа от 1 до 100, используя цикл do...until:

```
$i=1;
do {
   print $i . "<BR>\n";
   $i++;
} until ($i>100);
```

2.9.4. Цикл *foreach*

Цикл foreach используется для перебора элементов массива:

```
@Mass=qw(Один Два Три Четыре);
foreach $var (@Mass) {
   print "$var ";
} # Один Два Три Четыре
```

Если не указать переменную, то при каждой итерации цикла специальной переменной $_ будет присвоено значение элемента массива:

```
@Mass=qw(Один Два Три Четыре);
foreach (@Mass) {
   print "$_ ";
} # Один Два Три Четыре
```

Вместо массива можно указать диапазон значений. Выведем все числа от 1 до 100, используя цикл `foreach`:

```
foreach (1..100) {
   print "$_<BR>";
}
```

Перебрать элементы ассоциативного массива можно следующим образом:

```
%Mass = ("Один" => 1, "Два" => 2, "Три" => 3);
foreach $var (keys(%Mass)) {
   print "$var => $Mass{$var} ";
} # Выведет Два => 2 Три => 3 Один => 1
```

2.9.5. Оператор *next*

Оператор `next` позволяет перейти на следующую итерацию цикла еще до завершения выполнения всех выражений внутри цикла. Выведем все числа от 1 до 100, кроме чисел от 5 до 10:

```
for ($i=1; $i<101; $i++) {
  next if ($i>4 && $i<11);
  print $i . "<BR>\n";
}
```

2.9.6. Оператор *last*

Оператор `last` позволяет прервать выполнение цикла досрочно. Выведем все числа от 1 до 100, но при числе более 50 прервем цикл:

```
for ($i=1; $i<101; $i++) {
  last if ($i>50);
  print $i . "<BR>\n";
}
```

Оператор `last` прерывает выполнение цикла, а не программы, и далее будет выполнено выражение, следующее сразу за циклом.

2.9.7. Оператор *redo*

Оператор `redo` позволяет перейти к первому выражению в теле цикла еще до завершения выполнения всех выражений внутри цикла. В отличие от опера-

тора next не проверяется соответствие условию и не производится увеличение (или уменьшение) значения переменной-счетчика:

```
$i=1;
while ($i<101) {
   if ($i<131) {
      print $i . "<BR>\n";
      $i++;
      redo;
   }
}
```

В данном примере мы организовали цикл для вывода чисел от 1 до 100. Но внутри тела цикла мы используем оператор redo, который передаст управление циклу, только если переменная $i будет содержать значение больше 130. В этом случае проверяется условие цикла, и т. к. условие стало ложным, происходит выход из цикла. Иными словами, эта программа выведет все числа от 1 до 130, а не от 1 до 100.

2.9.8. Блоки

Фрагмент кода, заключенный в фигурные скобки, называется *блоком*. Каждый блок эквивалентен циклу, выполняемому один раз. По этой причине внутри блоков можно использовать операторы next, last и redo. Блок может быть с меткой и без метки. Если блок содержит метку, то он называется *именованным блоком*. Метка должна состоять из одного слова и заканчиваться двоеточием. В имени метки лучше использовать прописные буквы. С помощью блоков можно самостоятельно организовать цикл. Выведем все числа от 1 до 100, используя именованные блоки:

```
$i=1;
BLOCK: {
   last BLOCK if ($i>100);
   print $i . "<BR>\n";
   $i++;
   redo BLOCK;
}
```

В этой программе мы использовали оператор last для выхода из блока и оператор redo для возврата в начало блока. После этих операторов указывается идентификатор метки. Ранее мы использовали эти операторы без указания идентификатора метки, т. к. по умолчанию они управляют циклом, в котором вызваны. Иными словами, перед названиями циклов также можно указать метку. Исключение составляют циклы do...while и do...until:

```
BLOCK: for ($i=1; $i<101; $i++) {
  last BLOCK if ($i>50);
  print $i . "<BR>\n";
}
```

2.9.9. Оператор *goto*

С помощью оператора безусловного перехода `goto` можно передать управление в любое место программы. Например, возьмем предыдущий пример и вместо оператора `last` используем оператор `goto`:

```
for ($i=1; $i<101; $i++) {
  goto BLOCK if ($i>50);
  print $i . "<BR>\n";
}
BLOCK:;
```

Как и в операторе `last`, мы указываем идентификатор метки после оператора `goto`. Если переменная `$i` имеет значение больше 50, то мы передаем управление на строку с меткой `BLOCK`.

> ***Совет***
>
> Следует избегать использования оператора `goto`, его применение делает программу слишком запутанной и может привести к неожиданным результатам.

2.10. Функции. Разделяем программу на фрагменты

Функция (или процедура) — это фрагмент кода Perl, который можно вызвать из любого места программы. Функция описывается с помощью ключевого слова `sub` по следующей схеме:

```
sub <Имя функции> {
<Тело функции>
[return <Значение>]
}
```

Функция должна иметь уникальное имя, состоящее из букв, цифр и символа подчеркивания. Имя функции не может начинаться с цифры.

> ***Обратите внимание***
>
> Название функции зависит от регистра символов.

Между фигурными скобками располагаются выражения Perl. Кроме того, функция может возвращать значение в место вызова функции. Возвращаемое

значение задается с помощью ключевого слова `return`. Если ключевое слово `return` не указано, то возвращается значение последнего выражения.

Пример функции без параметров:

```
sub f_print_OK {
   print "Сообщение при удачно выполненной операции";
}
```

Пример функции с параметром:

```
sub f_print {
   $msg = shift();
   print $msg;
}
```

При вызове функции переданный параметр попадает в массив `@_`. Этот массив является параметром по умолчанию для большинства встроенных функций Perl. Для получения первого элемента массива мы используем функцию `shift()`. Вторую строку можно переписать так:

```
   $msg = shift(@_);
```

В итоге мы выводим переданный параметр в окно Web-браузера с помощью оператора `print`. Данную функцию можно переписать одной строкой кода:

```
sub f_print { print shift(); }
```

Пример функции с параметрами возвращающей сумму двух переменных:

```
sub f_Sum {
   ($x, $y) = @_;
   $z = $x + $y;
   return $z;
}
```

В данном случае мы получаем параметры из массива `@_` в список. Далее производим суммирование и возвращаем результат в место вызова функции с помощью ключевого слова `return`. В качестве возвращаемого значения `return` можно указывать не только имя переменной, но и выражение:

```
sub f_Sum {
   ($x, $y) = @_;
   return $x + $y;
}
```

В программе функции можно вызвать следующим образом:

```
f_print_OK();
f_print("Сообщение");
$var = f_Sum(5, 2); # Переменной $var будет присвоено значение 7
```

Кроме того, перед именем функции можно поместить идентификатор &:

```
&f_print_OK();
&f_print("Сообщение");
$var = &f_Sum(5, 2); # Переменной $var будет присвоено значение 7
```

Если мы не передаем функции никаких переменных, то скобки можно не указывать:

```
&f_print_OK;
```

> ***Совет***
>
> Всегда указывайте скобки при вызове функции. Это позволит сделать программу более наглядной.

Выражения, указанные после `return <Значение>;`, никогда не будут выполнены:

```
sub f_Sum {
   ($x, $y) = @_;
   return $x + $y;
   print "Сообщение"; # Это выражение никогда не будет выполнено
}
```

Некоторые параметры функции можно сделать необязательными. Например, сделаем второй параметр необязательным:

```
$var1=5;
$var3 = f_Sum($var1); # Переменной $var3 будет присвоено значение 7
$var4 = f_Sum($var1, 5); # Переменной $var4 будет присвоено значение 10

sub f_Sum {
   ($x, $y) = @_;
   $y=2 unless (defined($y));
   return $x + $y;
}
```

Если переменная `$y` не определена, то ей присваивается значение 2. Таким образом, если второй параметр не задан, то его значение будет равно 2.

2.10.1. Расположение функций

Функции могут располагаться и в любом месте программы или в отдельном файле в составе пакета или модуля:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

```
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

&f_print("Привет");
sub f_print { print shift(); }
```

2.10.2. Рекурсия. Вычисляем факториал

Рекурсия — это вызов функции самой себя. Это удобно, но если не предусмотреть условие выхода, приводит к бесконечным циклам. Для примера приведем вычисление факториала (листинг 2.34).

Листинг 2.34. Вычисление факториала

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

$fact = param("fact");
print "<HTML><HEAD>\n";
print "<TITLE>Вычисление факториала</TITLE>\n";
print "</HEAD><BODY>\n";
print "<B>Вычисление факториала</B><BR><BR>\n";
print "Введите число<BR>\n";
print "<FORM>\n";
print "<INPUT type=\"text\" name=\"fact\">\n";
print "<INPUT type=\"submit\" value=\"OK\">\n";
print "</FORM><BR>\n";
if (defined($fact)) {
   if ($fact !~ /^[0-9]+$/) {
      print "Необходимо ввести целое число!";
   }
   else {
      print "Факториал числа ", $fact, " = ", f_Factorial($fact);
   }
}
print "</BODY></HTML>\n";

sub f_Factorial {
   my $x = shift();
```

```
    if ($x == 0 || $x == 1) { return 1; }
    else {
      return $x * f_Factorial($x – 1);
    }
}
```

Обратите внимание на строку:

```
    my $x = shift();
```

Здесь мы описываем область видимости переменной с помощью ключевого слова `my`. Если этого не сделать, то функция в любом случае будет возвращать значение 1.

2.10.3. Глобальные, лексические и динамические переменные

Глобальные переменные — это переменные, которые видны в любой части программы и существуют на всем протяжении ее выполнения. Все переменные, не объявленные явным образом, являются глобальными переменными. Для явного объявления глобальной переменной используется ключевое слово `our`:

```
our $global = "Значение";
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Ключевое слово `our` появилось в версии Perl 5.6.

Лексические переменные — это переменные, объявленные внутри блока. Лексические переменные видны только внутри блока. Если в области действия лексической переменной существуют определения функций, то лексическая переменная будет видна внутри этих функций. Если имя лексической переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с лексической переменной, а значение глобальной не изменяется. Для объявления лексической переменной используется ключевое слово `my`:

```
my $var = "Значение";
```

Динамические переменные — это переменные, объявленные внутри блока. В отличие от лексических переменных динамические переменные видны не только внутри блока, но и во всех функциях, вызванных из этого блока. Если имя динамической переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с динамической переменной, а значение глобальной не изменяется. Для объявления динамической переменной используется ключевое слово `local`:

```
local $var = "Значение";
```

Рассмотрим все на примере (листинг 2.35).

Листинг 2.35. Глобальные, лексические и динамические переменные

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

our $var1 = "global";
our $var2 = "global";
our $var3 = "global";
my $var4 = "lexical";
print "<B>Начальные значения переменных:</B><BR>";
print "\$var1 = $var1<BR>";
print "\$var2 = $var2<BR>";
print "\$var3 = $var3<BR>";
print "\$var4 = $var4<BR>";
f_Function1(); # Вызываем функцию
print "<B>Конечные значения переменных:</B><BR>";
print "\$var1 = $var1<BR>";
print "\$var2 = $var2<BR>";
print "\$var3 = $var3<BR>";
print "\$var4 = $var4<BR>";
print "\$var5 = $var5<BR>";
print "\$var6 = $var6<BR>";

sub f_Function1 {
   $var1 .= " new";
   my $var2 = "lexical";
   local $var3 = "dynamic";
   $var4 .= " new";
   local $var5 = "dynamic";
   my $var6 = "lexical";
   print "<B>Значения переменных внутри функции f_Function1:</B><BR>";
   print "\$var1 = $var1<BR>";
   print "\$var2 = $var2<BR>";
   print "\$var3 = $var3<BR>";
   print "\$var4 = $var4<BR>";
   print "\$var5 = $var5<BR>";
```

```
   print "\$var6 = $var6<BR>";
   f_Function2();
}
sub f_Function2 {
   $var1 .= " vars";
   $var4 .= " vars";
   print "<B>Значения переменных внутри функции f_Function2:</B><BR>";
   print "\$var1 = $var1<BR>";
   print "\$var2 = $var2<BR>";
   print "\$var3 = $var3<BR>";
   print "\$var4 = $var4<BR>";
   print "\$var5 = $var5<BR>";
   print "\$var6 = $var6<BR>";
}
```

В окне Web-браузера получим следующий результат:

```
Начальные значения переменных:
$var1 = global
$var2 = global
$var3 = global
$var4 = lexical
Значения переменных внутри функции f_Function1:
$var1 = global new
$var2 = lexical
$var3 = dynamic
$var4 = lexical new
$var5 = dynamic
$var6 = lexical
Значения переменных внутри функции f_Function2:
$var1 = global new vars
$var2 = global
$var3 = dynamic
$var4 = lexical new vars
$var5 = dynamic
$var6 =
Конечные значения переменных:
$var1 = global new vars
$var2 = global
$var3 = global
$var4 = lexical new vars
$var5 =
$var6 =
```

Итак, рассмотрим каждую переменную по отдельности:

- `$var1`, объявленная как глобальная, видна во всех функциях и мы можем изменять ее значение внутри этих функций;
- `$var2` также объявлена как глобальная. Но внутри функции `f_Function1()` мы объявляем одноименную лексическую переменную и присваиваем ей значение. При вызове функции `f_Function2()` из `f_Function1()` лексическая переменная `$var2` не видна внутри функции `f_Function2()`, т. к. область видимости лексической переменной ограничена функцией, в которой она вызвана. По этой причине внутри функции `f_Function2()` мы получаем значение глобальной переменной `$var2`. Обратите внимание, присвоение лексической переменной нового значения внутри функции не изменило значение одноименной глобальной переменной;
- `$var3` изначально объявлена как глобальная. Но внутри функции `f_Function1()` мы объявляем одноименную динамическую переменную и присваиваем ей значение. При вызове функции `f_Function2()` из `f_Function1()` динамическая переменная `$var3` также видна и внутри функции `f_Function2()`, т. к. область видимости динамической переменной не ограничивается функцией, в которой она вызвана. Динамическая переменная видна во всех функциях, вызванных из первоначальной функции. По этой причине внутри функции `f_Function2()` мы получаем значение динамической переменной `$var3`. Обратите внимание, присвоение динамической переменной нового значения внутри функции не изменило значение одноименной глобальной переменной;
- `$var4` демонстрирует объявление лексической переменной вне функций. Так как в области действия лексической переменной существуют определения функций, то лексическая переменная будет видна внутри этих функций. По этой причине мы можем изменять ее значение внутри функций `f_Function1()` и `f_Function2()`;
- `$var5`, объявленная внутри функции `f_Function1()` как динамическая, видна внутри функции `f_Function2()`, но не видна вне этих функций;
- `$var6`, объявленная внутри функции `f_Function1()` как лексическая, не видна ни внутри функции `f_Function2()`, ни вне функций.

За один раз можно объявить несколько переменных, перечислив их внутри скобок:

```
our (@var, @mass, %hash);
my (@var, @mass, %hash);
local (@var, @mass, %hash);
```

2.10.4. Передача параметра по ссылке

Массивы и хеши можно передавать в функцию по ссылке. В этом случае создается переменная, которая не имеет собственного значения, а содержит ссылку на исходную переменную. Любые изменения, применяемые к ссылке, влияют и на исходную переменную. Иными словами, ссылка содержит не значение переменной, а ее адрес в памяти компьютера.

При передаче по ссылке перед именем переменной указывается знак \:

```
f_Function(\@mass); # Ссылка на массив
f_Function(\%hash); # Ссылка на хеш
```

Внутри функции к ссылке необходимо применить операцию разыменования. Для этого после идентификатора переменной указывается знак $:

```
@$mass_link # массив
%$hash_link # хеш
```

Для доступа к элементу массива или хеша используется следующий синтаксис:

```
$$mass_link[0] # массив
$$hash_link{'Ключ'} # хеш
```

В качестве примера рассмотрим передачу массива (листинг 2.36) и хеша (листинг 2.37) с помощью ссылок.

Листинг 2.36. Передача массива в функцию по ссылке

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

@mass = (1..10);
f_Function(\@mass);
foreach $var (@mass) {
  print "$var ";
} # Выведет 6 7 8 9 10 11 12 13 14 15

sub f_Function {
   my $mass_link = shift();
   foreach $var (@$mass_link) {
     $var += 5;
   }
}
```

Листинг 2.37. Передача хеша в функцию по ссылке

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

%hash = ("Один" => 1, "Два" => 2, "Три" => 3);
f_Function(\%hash);
while (($key, $value) = each(%hash)) {
   print "$key => $value ";
} # Выведет Два => 7 Три => 8 Один => 6
sub f_Function {
   my $hash_link = shift();
   foreach $var (keys (%$hash_link)) {
     $$hash_link{$var} += 5;
   }
}
```

2.10.5. Переменное число параметров в функции

По умолчанию все параметры, переданные функции, доступны через специальный массив `@_`. При использовании этого массива можно передать нашей функции произвольное количество параметров. Например, можно просуммировать сразу несколько чисел (листинг 2.38).

Листинг 2.38. Переменное число параметров в функции

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print f_Sum(5, 6, 7, 20); # Выведет 38
sub f_Sum {
   my $sum = 0;
   for (my $i=0; $i<@_; $i++) {
      $sum += $_[$i];
   }
   return $sum;
}
```

2.10.6. Функция *wantarray()*

Как вы уже знаете, с помощью ключевого слова `return` можно задать значение, возвращаемое функцией. Может быть указано скалярное значение или список. Функция `wantarray()` возвращает `true`, если функция вызвана из спискового контекста, и `false`, если из скалярного. Продемонстрируем это на примере (листинг 2.39).

Листинг 2.39. Функция `wantarray()`

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print "Скалярный контекст: " . f_Function() . "<BR>";
@mass2 = f_Function();
print "Списковый контекст: @mass2";
sub f_Function {
   my @mass = (1..10);
   if (wantarray()) {
      return @mass; # Списковый контекст
   }
   else {
      return scalar(@mass); # Скалярный контекст
   }
}
```

Вывод:

```
Скалярный контекст: 10
Списковый контекст: 1 2 3 4 5 6 7 8 9 10
```

2.11. Ссылки и сложные структуры данных

В предыдущих разделах мы познакомились с основными типами данных языка Perl: скалярными переменными, массивами и хешами. В этом разделе мы рассмотрим еще один скалярный тип данных — *ссылки*. При инициализации ссылки создается скалярная переменная, которая не имеет собственного значения, а содержит ссылку на исходную переменную. Иными словами,

ссылка содержит не значение переменной, а ее адрес в памяти компьютера. Создадим ссылку, а затем выведем сохраненный адрес:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

our $var1 = "Строка";
our $var2 = \$var1;
print $var2;
```

Вывод:

```
SCALAR(0x226cf8)
```

Чтобы получить значение переменной по этому адресу, необходимо произвести операцию *разыменования* ссылки. Для этого после идентификатора переменной указывается знак `$`:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

our $var1 = "Строка";
our $var2 = \$var1;
print $$var2;
```

Вывод:

```
Строка
```

Любые изменения, применяемые к ссылке, влияют и на исходную переменную. Для примера изменим значение переменной через ссылку:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

our $var1 = "Строка";
our $var2 = \$var1;
$$var2 = "Новая строка";
print $var1;
```

Вывод:

```
Новая строка
```

Рассмотрим различные типы ссылок более подробно.

2.11.1. Символические ссылки

Символическая ссылка — это скалярная переменная, которая содержит имя другой переменной. Причем имя переменной должно существовать в таблице имен пакета.

```
our $var = 12;
my $link = "var"; # Символическая ссылка
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Лексические переменные, объявленные с помощью ключевого слова `my`, не помещаются в таблицу имен пакета. По этой причине нельзя создать символическую ссылку на лексическую переменную.

Чтобы получить значение переменной, расположенной по ссылке, необходимо произвести операцию разыменования. Для этого перед именем ссылки указываются два знака `$`:

```
$$link
```

При этом имя ссылки можно заключить в фигурные скобки:

```
${$link}
```

Если в программе включена прагма `strict`, то использование символических ссылок приводит к фатальному завершению программы. Для использования символических ссылок следует отключить проверку использования ссылок с помощью выражения:

```
no strict 'refs';
```

В листинге 2.40 приведен пример создания и разыменования символических ссылок.

Листинг 2.40. Использование символических ссылок

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

no strict 'refs';

our $var1 = 10;
my $link1 = "var1"; # Символическая ссылка
print "Значение переменной \$var1 = ", $$link1, "<BR>";
print "Значение переменной \$var1 = ", ${$link1}, "<BR>";
```

```
my $var2 = 5;
my $link2 = "var2"; # Символическая ссылка
print "Значение переменной \$var2 = ", $$link2;
```

Вывод:

```
Значение переменной $var1 = 10
Значение переменной $var1 = 10
Значение переменной $var2 =
```

В последнем случае мы не получили значения переменной `$var2`, т. к. она объявлена в программе с помощью ключевого слова `my`.

2.11.2. Жесткие ссылки

В отличие от символических ссылок *жесткие ссылки* указывают не на имя переменной, а на ее адрес в памяти компьютера. Жесткие ссылки используются гораздо чаще символических ссылок. В дальнейшем под словом "ссылка" мы будем подразумевать именно жесткую ссылку.

Для создания ссылки на скалярную переменную необходимо перед переменной указать обратный слэш:

```
my $var = 10;
my $link = \$var;
```

Чтобы получить значение переменной, расположенной по ссылке, необходимо произвести операцию разыменования. Для этого перед именем ссылки указываются два знака `$`:

```
$$link
```

В листинге 2.41 приведен пример создания и разыменования ссылок на скалярные переменные.

Листинг 2.41. Использование жестких ссылок

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $var = 10;
my $link = \$var;
$$link++;
print "Значение переменной \$var = ", $var, "<BR>";
print "Значение переменной \$var = ", $$link, "<BR>";
print "Значение переменной \$var = ", ${$link};
```

Вывод:

```
Значение переменной $var = 11
Значение переменной $var = 11
Значение переменной $var = 11
```

Как вы уже знаете, если лексическая переменная объявлена внутри блока, то после выхода из блока она удаляется и больше не доступна программе. Если внутри блока создана ссылка на эту переменную, то значение переменной не удаляется и доступно через разыменование ссылки. В то же время, если после блока попытаться вывести значение лексической переменной обычным образом, значения мы не получим:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

my $link;
{
   my $var = 10;
   $link = \$var;
}
print "Переменная \$var = ", $var, "<BR>";
print "Значение ссылки ", $$link;
```

Вывод:

```
Переменная $var =
Значение ссылки 10
```

Функция `ref()` для ссылок на скалярные переменные выводит значение `SCALAR`:

```
my $var = 10;
my $link = \$var;
print ref($link); # Выведет SCALAR
```

Кроме создания ссылок на скалярные переменные мы можем создавать ссылки на ссылки:

```
my $var = 10;
my $link1 = \$var;
my $link2 = \$link1;
```

Чтобы получить значение исходной переменной, следует разыменовать ссылку следующим образом:

```
$$$link2
```

Функция `ref()` для ссылок на ссылки выводит значение `REF`:

```
my $var = 10;
my $link1 = \$var;
my $link2 = \$link1;
print ref($link2); # Выведет REF
```

2.11.3. Ссылки на массивы

При создании ссылки на массив необходимо перед массивом указать обратный слэш:

```
my @mass = ("Один", "Два", "Три", "Четыре");
my $link = \@mass;
```

Разыменовать ссылку на массив можно двумя способами:

- указав два знака `$`:

```
$$link[0]
```

- используя оператор -> ("стрелка"):

```
$link->[1]
```

Для разыменования ссылки на весь массив следует указать символы `@$` перед именем ссылки:

```
foreach my $var (@$link) {
   print "$var ";
}
```

В листинге 2.42 приведен пример создания и разыменования ссылок на массивы.

Листинг 2.42. Ссылки на массивы

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my @mass = ("Один", "Два", "Три", "Четыре");
my $link = \@mass;
print $$link[0], "<BR>";
print $link->[1], "<BR>";
foreach my $var (@$link) {
   print "$var ";
}
```

Вывод:

```
Один
Два
Один Два Три Четыре
```

Ссылку на массив можно создать с помощью анонимных структур. В этом случае для создания ссылки необходимо заключить список значений в квадратные скобки:

```
my $link = ["Один", "Два", "Три", "Четыре"];
```

Разыменование ссылки на анонимный массив производится аналогично разыменованию ссылок на обычные массивы:

```
$$link[0]
$link->[1]
@$link
```

При желании, мы можем добавить новый элемент массива или изменить значение существующего элемента:

```
$link->[4] = "Пять";
```

В листинге 2.43 приведен пример создания и разыменования ссылок на анонимные массивы.

Листинг 2.43. Ссылки на анонимные массивы

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $link = ["Один", "Два", "Три", "Четыре"];
$link->[4] = "Пять";
print $$link[0], "<BR>";
print $link->[1], "<BR>";
foreach my $var (@$link) {
   print "$var ";
}
```

Вывод:

```
Один
Два
Один Два Три Четыре Пять
```

Функция `ref()` для ссылок на массивы выводит значение `ARRAY`:

```
my $link = ["Один", "Два", "Три", "Четыре"];
print ref($link); # Выведет ARRAY
```

2.11.4. Ссылки на ассоциативные массивы

При создании ссылки на ассоциативный массив необходимо перед ассоциативным массивом указать обратный слэш:

```
my %hash = ("Один" => 1, "Два" => 2, "Три" => 3);
my $link = \%hash;
```

Разыменовать ссылку на хеш можно двумя способами:

- указав два знака `$`:

```
$$link{'Один'}
```

- используя оператор `->`:

```
$link->{'Два'}
```

Для разыменования ссылки на весь хеш следует указать символы `%$` перед именем ссылки:

```
while (my ($key, $value) = each(%$link)) {
   print "$key => $value ";
}
```

В листинге 2.44 приведен пример создания и разыменования ссылок на хеши.

Листинг 2.44. Ссылки на хеши

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my %hash = ("Один" => 1, "Два" => 2, "Три" => 3);
my $link = \%hash;
print $$link{'Один'}, "<BR>";
print $link->{'Два'}, "<BR>";
while (my ($key, $value) = each(%$link)) {
   print "$key => $value ";
}
```

Вывод:

```
1
2
Два => 2 Три => 3 Один => 1
```

Ссылку на ассоциативный массив можно создать с помощью анонимных структур. В этом случае для создания ссылки необходимо заключить список значений в фигурные скобки:

```
my $link = {"Один" => 1, "Два" => 2, "Три" => 3};
```

Разыменование ссылки на анонимный хеш производится аналогично разыменованию ссылок на обычные хеши:

```
$$link{'Один'}
$link->{'Два'}
%$link
```

Мы можем добавить новый элемент хеша или изменить значение существующего элемента:

```
$link->{"Четыре"} = 4;
```

В листинге 2.45 приведен пример создания и разыменования ссылок на анонимные хеши.

Листинг 2.45. Ссылки на анонимные хеши

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $link = {"Один" => 1, "Два" => 2, "Три" => 3};
$link->{"Четыре"} = 4;
print $$link{'Один'}, "<BR>";
print $link->{'Два'}, "<BR>";
while (my ($key, $value) = each(%$link)) {
   print "$key => $value ";
}
```

Вывод:

```
1
2
Четыре => 4 Два => 2 Три => 3 Один => 1
```

Функция `ref()` для ссылок на хеши выводит значение `HASH`:

```
my $link = {"Один" => 1, "Два" => 2, "Три" => 3};
print ref($link); # Выведет HASH
```

2.11.5. Ссылки на функции

Можно создать ссылку на функцию:

```
my $link = \&f_print_OK;
sub f_print_OK {
   print "Сообщение при удачно выполненной операции<BR>";
}
```

Разыменовать ссылку на функцию можно двумя способами:

- указав знак `&`:

```
&$link()
```

- используя оператор `->`:

```
$link->()
```

В листинге 2.46 приведен пример создания и разыменования ссылок на функции.

Листинг 2.46. Ссылки на функции

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $link = \&f_print_OK;
&$link();
$link->();

sub f_print_OK {
   print "Сообщение при удачно выполненной операции<BR>";
}
```

Вывод:

```
Сообщение при удачно выполненной операции
Сообщение при удачно выполненной операции
```

Ссылку на функцию можно создать с помощью анонимных структур. Создание ссылки производится следующим образом:

```
my $link = sub
{
   print "Сообщение при удачно выполненной операции<BR>";
};
```

Разыменование ссылки на анонимную функцию производится следующим образом:

```
&$link();
$link->();
```

В листинге 2.47 приведен пример создания и разыменования ссылки на анонимную функцию.

Листинг 2.47. Ссылки на анонимные функции

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $link = sub
{
   print "Сообщение при удачно выполненной операции<BR>";
};
&$link();
$link->();
```

Вывод:

```
Сообщение при удачно выполненной операции
Сообщение при удачно выполненной операции
```

Внутри обычной функции мы можем создать анонимную функцию и возвратить ссылку на нее. Такой метод возврата значения называется *замыканием*.

> ***ОБРАТИТЕ ВНИМАНИЕ***
> Замыкания возможны только при использовании лексических переменных.

При каждом вызове функции с замыканием сохраняется дополнительная версия значений лексических переменных. Продемонстрируем это на примере (листинг 2.48).

Листинг 2.48. Использование замыканий

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
```

```
use strict;
print "Content-type: text/html\n\n";

sub f_print {
   my $txt1 = shift;
   return sub { # Возвращаем замыкание
      my $txt2 = shift;
      print $txt1, $txt2;
   }
}

my $link1 = &f_print("Привет, "); # Замыкание1
my $link2 = &f_print("Прощай, "); # Замыкание2

$link1->("Николай"); # Разыменование1
print "<BR>";
&$link2("Сергей"); # Разыменование2
```

Вывод:

```
Привет, Николай
Прощай, Сергей
```

С помощью создания и разыменования ссылки на функцию мы можем вставить значение, возвращаемое функцией, в строку, заключенную в кавычки. Для примера рассмотрим функцию суммирования двух чисел:

```
sub f_Sum {
   return ($_[0] + $_[1]);
}
```

Чтобы вывести значение функции в строку, обычно используется следующий синтаксис:

```
print "2 + 2 = ", f_Sum(2, 2);
```

Если название функции окажется внутри кавычек, то будет выведено название функции, а не ее значение:

```
print "2 + 2 = f_Sum(2, 2)";
```

Вывод:

```
2 + 2 = f_Sum(2, 2)
```

Чтобы получить значение, необходимо создать ссылку на функцию, а затем ее разыменовать:

```
print "2 + 2 = ${ \f_Sum(2, 2) }";
```

Вывод:

```
2 + 2 = 4
```

Функция `ref()` для ссылок на функции выводит значение `CODE`:

```
sub f_print_OK {
   print "Сообщение при удачно выполненной операции<BR>";
}
my $link = \&f_print_OK;
print ref($link); # Выведет CODE
```

2.11.6. Многомерные массивы

Так как ссылка имеет скалярный тип данных, мы можем сохранять ссылку как элемент массива. Таким образом можно создавать очень сложные структуры данных. Для примера создадим двумерный массив:

```
my @mass1 = ("Один", "Два", "Три", "Четыре");
my @mass2 = ("Пять", "Шесть", "Семь", "Восемь");
my @mass_ref = (\@mass1, \@mass2);
```

Последнюю строку можно заменить:

```
my @mass_ref = \(@mass1, @mass2);
```

Создать такой же двумерный массив ссылок можно с помощью анонимных структур:

```
my @mass_ref = (
      ["Один", "Два", "Три", "Четыре"],
      ["Пять", "Шесть", "Семь", "Восемь"]
);
```

Получить доступ к элементу такого массива ссылок можно одним из следующих способов:

- указав индексы в квадратных скобках:

```
$mass_ref[1][1]
```

- указав между первым и вторым индексами оператор `->`:

```
$mass_ref[1]->[1]
```

- указав символ `$`:

```
${$mass_ref[1]}[1]
```

Вывести все элементы такого массива ссылок можно таким образом:

```
foreach my $row (@mass_ref) {
   foreach my $item (@$row) {
```

```
      print $item, " ";
   }
   print "<BR>";
}
```

Или таким:

```
for(my $i=0; $i<@mass_ref; $i++) {
   for(my $j=0; $j<@{$mass_ref[$i]}; $j++) {
      print $mass_ref[$i][$j], " ";
   }
   print "<BR>";
}
```

При желании можно добавить новые элементы такого массива. Добавим две строки и выведем результат:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my @mass_ref = (
      ["Один", "Два", "Три", "Четыре"],
      ["Пять", "Шесть", "Семь", "Восемь"]
);
$mass_ref[2] = ["Девять", "Десять"];
my @mass = ("Одиннадцать", "Двенадцать");
@{$mass_ref[3]} = @mass;

for(my $i=0; $i<@mass_ref; $i++) {
   for(my $j=0; $j<@{$mass_ref[$i]}; $j++) {
      print $mass_ref[$i][$j], " ";
   }
   print "<BR>";
}
```

Обратите внимание на строку:

```
@{$mass_ref[3]} = @mass;
```

Напишем так:

```
$mass_ref[3] = @mass;
```

Элементу массива будет присвоено количество элементов массива `@mass`, а не сами элементы массива.

Кроме массива ссылок можно создать ссылку на массив:

```
my $link = [
      ["Один", "Два", "Три", "Четыре"],
      ["Пять", "Шесть", "Семь", "Восемь"]
];
```

Получить доступ к элементу такой структуры можно одним из следующих способов:

- указав оператор -> после имени ссылки, а затем два индекса:

```
$link->[1][1]
```

- указав между именем ссылки, первым и вторым индексами оператор ->:

```
$link->[1]->[1]
```

- указав символ $:

```
$$link[1][1]
```

Вывести все элементы такой структуры можно следующим образом:

```
foreach my $row (@$link) {
   foreach my $item (@$row) {
      print $item, " ";
   }
   print "<BR>";
}
```

Или таким:

```
for(my $i=0; $i<@$link; $i++) {
   for(my $j=0; $j<@{$link->[$i]}; $j++) {
      print $link->[$i][$j], " ";
   }
   print "<BR>";
}
```

При желании можно добавить новые элементы такой структуры. Для примера добавим две строки и выведем результат:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my $link = [
      ["Один", "Два", "Три", "Четыре"],
```

```
      ["Пять", "Шесть", "Семь", "Восемь"]
];
$link->[2] = ["Девять", "Десять"];
my @mass = ("Одиннадцать", "Двенадцать");
$link->[3] = [@mass];

for(my $i=0; $i<@$link; $i++) {
   for(my $j=0; $j<@{$link->[$i]}; $j++) {
      print $link->[$i][$j], " ";
   }
   print "<BR>";
}
```

Обратите внимание на строку:

```
$link->[3] = [@mass];
```

Если опустить квадратные скобки:

```
$link->[3] = @mass; # Неправильно
```

Элементу структуры будет присвоено количество элементов массива `@mass`, а не сами элементы массива.

2.11.7. Многомерные хеши

Создать хеш анонимных хешей можно следующим образом:

```
my %tag = (
      "p" => {
         "text" => "Текст параграфа",
         "align" => "center"
       },
      "img" => {
         "src" => "foto.gif",
         "align" => "left",
         "hspace" => "20",
         "vspace" => "10"
       }
);
```

Получить значение элемента такой структуры можно одним из следующих способов:

```
$tag{"img"}{"src"}
$tag{"img"}->{"src"}
${$tag{"img"}}{"src"}
```

При желании можно добавить еще один анонимный хеш к нашей структуре:

```
$tag{"a"} = {
      "href" => "file.html",
      "target" => "_blank"
};
```

Для вывода всех элементов структуры в отсортированном виде можно воспользоваться следующим способом:

```
foreach my $var1 (sort(keys(%tag))) {
   print "<B>$var1:</B> ";
   foreach my $var2 (sort(keys(%{$tag{"$var1"}}))) {
      print "$var2 => $tag{$var1}{$var2}; ";
   }
   print "<BR>";
}
```

2.11.8. Массивы хешей

Создать массив хешей можно следующим образом:

```
my @car = (
      {
         "model" => "ВАЗ 2107",
         "year" => "2003",
         "price" => "1300",
         "color" => "красный"
       },
      {
         "model" => "ВАЗ 2109",
         "year" => "2006",
         "price" => "2500",
         "color" => "белый"
       }
);
```

Получить значение элемента такой структуры можно одним из следующих способов:

```
$car[0]{"model"}
$car[0]->{"model"}
${$car[0]}{"model"}
```

При желании можно добавить еще один элемент массива:

```
push(@car,
{
         "model" => "Москвич 412",
```

```
        "year" => "1980",
        "price" => "150",
        "color" => "синий"
}
);
```

Для вывода всех элементов массива хешей в отсортированном виде можно воспользоваться следующим способом:

```
foreach my $var (@car) {
   foreach my $key (sort(keys(%$var))) {
      print "$key => $var->{$key}; ";
   }
   print "<BR>";
}
```

Или с помощью цикла `for`:

```
for (my $i=0; $i<@car; $i++) {
   print "<B>[$i]:</B> ";
   foreach my $key (sort(keys(%{$car[$i]}))) {
      print "$key => $car[$i]{$key}; ";
   }
   print "<BR>";
}
```

2.11.9. Хеши массивов

Создать хеш массивов можно следующим образом:

```
my %word = (
      "Слово1" => ["Определение1", "Определение2", "Определение3"],
      "Слово2" => ["Определение1", "Определение2"]
);
```

Получить значение элемента такой структуры можно одним из следующих способов:

```
$word{"Слово1"}[2]
$word{"Слово1"}->[2]
${$word{"Слово1"}}[2]
```

При желании можно добавить еще один элемент хеша:

```
$word{"Слово3"} = ["Определение1", "Определение2"];
```

Для вывода всех элементов хеша массивов можно воспользоваться следующим способом:

```
foreach my $key (sort(keys(%word))) {
   print "$key => ";
   for (my $i=0; $i<@{$word{$key}}; $i++) {
      print "$word{$key}[$i] ";
   }
   print "<BR>";
}
```

2.11.10. Хеши функций

В некоторых языках программирования (например, в PHP) существует оператор переключения `switch`. Оператор имеет следующий формат:

```
switch (<Переменная или выражение>) {
case <Значение 1>:
<Выражение 1>;
case <Значение 2>:
<Выражение 2>;
...
default:
<Выражение>;
}
```

В зависимости от значения переменной или выражения выполняется тот или иной блок `case`. Если значение отсутствует в блоках `case`, то выполняется блок `default`. В языке Perl такого оператора не существует, но его можно эмулировать с помощью хеша функций. Продемонстрируем это на примере (листинг 2.49).

Листинг 2.49. Хеши функций

```
#!/usr/bin/perl -w
use CGI qw( :standard);
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

print "Введите значение<BR>\n";
print "<FORM>\n";
print "<INPUT type=\"text\" name=\"var\">\n";
print "<INPUT type=\"submit\" value=\"OK\">\n";
print "</FORM>\n";

my %switch = (
         "Значение1" => \&f_sub1,
```

```
        "Значение2" => \&f_sub2,
        "Значение3" => \&f_sub3,
        "Значение4" => \&f_sub4
);
my $str = param("var");

if ($str eq "") {
   print "Необходимо ввести значение";
}
elsif (exists($switch{$str})) {
   $switch{$str}->();
}
else {
   print "Такого значения нет";
}

sub f_sub1 {
   print "Значение1";
}
sub f_sub2 {
   print "Значение2";
}
sub f_sub3 {
   print "Значение3";
}
sub f_sub4 {
   print "Значение4";
}
```

Теперь в зависимости от введенного значения будет выполняться соответствующая функция. Если значения нет в хеше `%switch`, то выводится сообщение "Такого значения нет". Если поле не заполнено, то выводится сообщение "Необходимо ввести значение".

2.11.11. Сложные структуры данных

До сих пор мы рассматривали двухуровневые структуры данных. Но структуры в Perl не ограничиваются двумя уровнями и однородностью данных. Например, мы можем создать массив, первый элемент которого является простой строкой, второй элемент — хешем, третий — массивом, четвертый — ссылкой на функцию и т. д. Для примера опишем следующую структуру данных:

```
<A href="file.html" target="_blank" alt="Подсказка"
style="font-size: 12pt; color: red; font-family: Arial">
```

Первый элемент нашего массива будет содержать название тега. Второй элемент будет содержать ассоциативный массив параметров тегов. Кроме того, в виде ассоциативного массива опишем атрибуты стиля в параметре `style`:

```
my @tag = ( "A",
      {
         "href" => "file.html",
         "target" => "_blank",
         "alt" => "Подсказка",
         "style" =>
            {
               "font-size" => "12pt",
               "color" => "red",
               "font-family" => "Arial"
            }
      }
);
```

Выведем значение атрибута `color` параметра `style`:

```
print $tag[1]{"style"}{"color"};
```

А теперь просто название тега:

```
print $tag[0];
```

Вывести все элементы такой структуры данных позволяет следующий код:

```
for (my $i=0; $i<@tag; $i++) {
   if (ref($tag[$i]) eq 'HASH') {
      print "<B>Параметры тега:</B><BR>";
      foreach my $key (sort(keys(%{$tag[$i]}))) {
         if (ref($tag[$i]{$key}) eq 'HASH') {
            print "<B>$key</B>";
            foreach my $item (sort(keys(%{$tag[$i]{$key}}))) {
               print "<LI>$item => ";
               print $tag[$i]{$key}{$item}, "<BR>";
            }
         }
         else {
            print "$key => ";
            print $tag[$i]{$key}, "<BR>";
         }
      }
   }
   else {
      print "Название тега — ";
```

```
      print $tag[$i], "<BR>";
   }
}
```

2.11.12. Тип данных *typeglob*

Тип данных `typeglob` служит для ссылки на все типы данных, к которым может относиться имя глобальной переменной. Все имена переменных берутся из таблицы имен пакета.

> ***Обратите внимание***
> Лексические переменные не помещаются в таблицу имен пакета.

В качестве идентификатора для типа `typeglob` используется символ `*`. Предположим, у нас определены три глобальных переменных разных типов с одинаковыми именами и одноименная функция:

```
our $var = "Строка";
our @var = ("Элемент1", "Элемент2");
our %var = ("Один" => "1", "Два" => "2");
sub var {
   return "Фунция var";
}
```

На все эти имена ссылается переменная `*var`. Чтобы получить ссылку на определенный тип данных, можно воспользоваться следующим кодом:

```
my $varref1 = *var{SCALAR}; # то же самое, что $varref1 = \$var
my $varref2 = *var{ARRAY}; # то же самое, что $varref2 = \@var
my $varref3 = *var{HASH}; # то же самое, что $varref3 = \%var
my $varref4 = *var{CODE}; # то же самое, что $varref4 = \&var
```

В дальнейшем можно разыменовать ссылку и вывести значение переменной:

```
print $$varref1, "<BR>\n";
print @$varref2, "<BR>\n";
print %$varref3, "<BR>\n";
print &$varref4, "<BR>\n";
```

С помощью типа `typeglob` можно создать константу:

```
*PI = \3.14;
print $PI;
```

Дальнейшее изменение этой константы приводит к ошибке "Modification of a read-only value attempted".

Функция `ref()` для ссылки на тип данных `typeglob` возвращает значение `GLOB`:

```
my $link = \*var;
print ref($link); # Выведет GLOB
```

2.12. Пакеты и модули

Пакет — это пространство имен для отдельного фрагмента программы. Каждый пакет содержит таблицу имен всех переменных и функций в виде хеша. По существу *модуль* — это тот же пакет, только обладающий дополнительными свойствами. Все созданные нами программы были частью пакета с названием `main`. Для обращения к переменной мы просто указывали имя переменной после идентификатора. Обратиться к переменной можно, указав название пакета. Например, к переменной `$var` можно обратиться следующим образом:

```
$main::var
$::var
```

В последнем случае мы не указали название пакета, т. к. пакет `main` подразумевается по умолчанию.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Лексические переменные не помещаются в таблицу имен пакета, а любая глобальная переменная на самом деле является глобальной переменной пакета.

Чтобы узнать имя пакета, можно воспользоваться лексемой `__PACKAGE__`:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

our $var1 = 10;
my $var2 = 5;
local $var3 = 8;
print "Имя пакета: " . __PACKAGE__ . "<BR>";
print "Глобальная переменная \$var1 = " . $main::var1 . "<BR>";
print "Лексическая переменная \$var2 = " . $main::var2 . "<BR>";
print "Динамическая переменная \$var3 = " . $main::var3 . "<BR>";
```

Вывод:

```
Имя пакета: main
Глобальная переменная $var1 = 10
```

```
Лексическая переменная $var2 =
Динамическая переменная $var3 = 8
```

Как видно из примера, мы не получили значения лексической переменной. Для обращения к функции пакета используется следующий синтаксис:

```
&<Имя пакета>::<Название функции>[()]
```

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

print &main::f_Sum(5, 6, 7, 20); # Выведет 38
sub f_Sum {
   my $sum = 0;
   for (my $i=0; $i<@_; $i++) {
      $sum += $_[$i];
   }
   return $sum;
}
```

2.12.1. Создание пакета

До сих пор мы использовали название пакета по умолчанию. Для создания пакета с другим именем применяется ключевое слово `package`:

```
package <Название пакета>;
```

Для примера создадим пакет с именем `First` (листинг 2.50).

Листинг 2.50. Создание пакета

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

our $var1 = 10;
print "Имя пакета: " . __PACKAGE__ . "<BR>";
print "Переменная \$var1 = " . $var1 . "<BR><BR>";

package First;
our $var1 = 5;
```

```
print "Имя пакета: " . __PACKAGE__ . "<BR>";
print "Переменная \$var1 = " . $var1 . "<BR>";
print "Переменная \$main::var1 = " . $main::var1 . "<BR>";
print "Переменная \$First::var1 = " . $First::var1 . "<BR>";
```

Вывод:

```
Имя пакета: main
Переменная $var1 = 10

Имя пакета: First
Переменная $var1 = 5
Переменная $main::var1 = 10
Переменная $First::var1 = 5
```

Вначале мы создаем глобальную переменную `$var1` и выводим название пакета и значение переменной. Далее с помощью ключевого слова `package` мы создаем пакет `First` и объявляем глобальную переменную с таким же именем, но другим значением. Выводим значение переменной `$var1` без указания пакета и получаем значение переменной из пакета `First`. Следующая строка демонстрирует обращение к переменной другого пакета. Как нетрудно заметить, значение переменной `$var1` из пакета `main` не изменилось после объявления одноименной переменной в пакете `First`. Последняя строка демонстрирует доступ к переменной с указанием названия пакета.

2.12.2. Выносим пакет в отдельный файл. Создаем шаблоны для множества страниц

Если пакет вынесен в отдельный файл, то подключить его позволяет оператор `require`. При использовании оператора `require` пакет подключается во время выполнения программы, а не при компиляции. Оператор имеет следующий формат:

```
require <Имя файла>;
require "<Имя файла>";
require <Номер версии>;
```

По умолчанию подключаемые файлы должны иметь расширение pm. Если файл имеет именно это расширение, то можно указать только имя файла:

```
require First;
require "First.pm";
```

Оператор `require` позволяет использовать другие расширения файла, например, pl. Если имя файла указывается с расширением, то название файла необ-

ходимо обязательно заключить в кавычки. В противном случае будет ошибка, т. к. точка является оператором конкатенации двух строк:

```
require "First.pl";
```

По умолчанию интерпретатор ищет подключаемый файл в каталогах, указанных в массиве `@INC`. Выведем текущее содержимое этого массива:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

for (my $i=0; $i<@INC; $i++) {
   print $INC[$i], "<BR>";
}
```

Вывод:

```
Z:/usr/local/perl/site/lib
Z:/usr/local/perl/lib
.
```

Символ точки означает, что файл может находиться в той же папке, что и исполняемый файл. При желании можно добавить свой каталог в массив `@INC` с помощью функции `unshift()` (в начало массива) или `push()` (в конец массива):

```
unshift(@INC, "Z:/home/perlbook.ru");
for (my $i=0; $i<@INC; $i++) {
   print $INC[$i], "<BR>";
}
```

Вывод:

```
Z:/home/perlbook.ru
Z:/usr/local/perl/site/lib
Z:/usr/local/perl/lib
.
```

Вынесем функцию `f_Sum()` в отдельный файл (листинг 2.51) и подключим его с помощью оператора `require` (листинг 2.52).

Листинг 2.51. Содержимое файла Sum.pl

```
package Sum;
sub f_Sum {
```

```
   my ($x, $y) = @_;
   return ($x + $y);
}
return 1; # означает, что файл успешно подключен
```

Листинг 2.52. Использование оператора `require`

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

require "Sum.pl";
my $var1=5;
my $var2=25;
print &Sum::f_Sum($var1, $var2);
```

Если подключаемый файл не найден, то будет выведено сообщение об ошибке:

```
Can't locate Sum.pl in @INC
```

С помощью подключаемых файлов можно создавать шаблоны для множества страниц сайта. Разместим HTML-код в подключаемом файле (листинг 2.53), а затем подключим его к основному файлу (листинг 2.54).

Листинг 2.53. Содержимое файла allscript.pm

```
package allscript;
sub f_Sum {
   my ($x, $y) = @_;
   return ($x + $y);
}
sub f_Header { # Верхний колонтитул
   print <<METKA;
<HTML>
<HEAD>
<TITLE>Функции</TITLE>
</HEAD>
<BODY>
METKA
}
sub f_Footer { # Нижний колонтитул
   print <<METKA;
```

```
</BODY>
</HTML>
METKA
}
return 1; # означает, что файл успешно подключен
```

Листинг 2.54. Размещение HTML-кода в подключаемом файле

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

require allscript;
&allscript::f_Header(); # Выводим верхний колонтитул
my $var1=5;
my $var2=25;
print &allscript::f_Sum($var1, $var2);
&allscript::f_Footer(); # Выводим нижний колонтитул
```

В итоге исходный HTML-код будет выглядеть следующим образом:

```
<HTML>
<HEAD>
<TITLE>Функции</TITLE>
</HEAD>
<BODY>
30</BODY>
</HTML>
```

2.12.3. Оператор *use*

Оператор `use` применяется для подключения модулей. При использовании оператора `use` модуль подключается во время компиляции, а не во время выполнения программы. Оператор имеет следующий формат:

```
use <Имя файла>;
use <Имя файла> <Минимальный номер версии модуля>;
use <Минимальный номер версии интерпретатора Perl>;
```

В отличие от оператора `require` оператор `use` позволяет подключать файлы только с расширением pm. Если указать другое расширение, то при компиляции будет выведено сообщение о неисправимой ошибке, и программа выполняться не будет. При подключении модуля можно указать минимальный но-

мер версии модуля, а также минимальную версию самого интерпретатора. Для указания номера версии модуля необходимо внутри модуля объявить глобальную переменную $VERSION:

```
our $VERSION = 2.0;
```

При подключении модуля в операторе use указываем минимальный номер версии:

```
use allscript 1.0;
```

Если версия модуля меньше требуемой, то при компиляции будет выведено сообщение о неисправимой ошибке, и программа выполняться не будет. Для правильной работы модуля можно также указать минимальную версию интерпретатора:

```
use v5.8.8;
```

Вынесем функцию f_Sum() в отдельный модуль (листинг 2.55) и подключим его с помощью оператора use (листинг 2.56).

Листинг 2.55. Содержимое файла Sum.pm

```
package Sum;
sub f_Sum {
   my ($x, $y) = @_;
   return ($x + $y);
}
return 1; # означает, что файл успешно подключен
```

Листинг 2.56. Использование оператора use

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use Sum;
my $var1=5;
my $var2=25;
print &Sum::f_Sum($var1, $var2);
```

Для обращения к функции модуля используется следующий синтаксис:

```
&<Имя модуля>::<Название функции>[()]
```

А для получения значения скалярной переменной модуля:

```
$<Имя модуля>::<Имя скалярной переменной>
```

Строку:

```
return 1;
```

следует обязательно указывать в конце модуля, т. к. последнее выражение модуля должно возвращать значение `true`.

По умолчанию интерпретатор ищет модули в каталогах, указанных в массиве `@INC`. При использовании оператора `require` достаточно было добавить новый путь с помощью функций `unshift()` и `push()`. При подключении модуля с помощью оператора `use` этого недостаточно, т. к. оператор `use` подключает модули на этапе компиляции, а не во время выполнения программы. По этой причине добавление нового пути будет произведено уже после попытки подключения модуля, и мы получим сообщение об ошибке, если модуль будет не найден. Для добавления нового пути необходимо использовать прагму `lib`:

```
use lib "Z:/home/perlbook.ru";
# Далее подключаем наш модуль
```

2.12.4. Экспортирование идентификаторов из модуля

При использовании модулей можно экспортировать идентификаторы из модуля в пространство имен другого пакета. Это достигается подключением модуля `Exporter`:

```
use Exporter;
our @ISA = qw( Exporter );
```

Далее заполняем массив `@EXPORT` экспортируемыми идентификаторами:

```
our @EXPORT = qw( &f_Sum );
```

В этой строке мы экспортировали название функции `f_Sum()`. Теперь к функции можно обратиться напрямую без указания названия модуля:

```
&f_Sum($var1, $var2);
```

В качестве примера вынесем функцию `f_Sum()` в отдельный модуль (листинг 2.57) и подключим его с помощью оператора `use` (листинг 2.58).

Листинг 2.57. Содержимое файла Sum.pm

```
package Sum;
use Exporter;
our @ISA = qw( Exporter );
```

```
our @EXPORT = qw( &f_Sum );
sub f_Sum {
   my ($x, $y) = @_;
   return ($x + $y);
}
return 1; # означает, что файл успешно подключен
```

Листинг 2.58. Экспортирование идентификаторов

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use Sum;
my $var1=5;
my $var2=25;
print &f_Sum($var1, $var2);
```

Кроме того, в массив `@EXPORT_OK` можно добавить идентификаторы, которые будут экспортироваться только при явном указании их в списке импорта:

```
use <Имя файла> qw(<Импортируемые идентификаторы>);
```

Это выражение позволяет импортировать только указанные идентификаторы, а не те, что заданы в массиве `@EXPORT`.

В качестве примера создадим модуль `Func.pm` (листинг 2.59) и явно импортируем идентификаторы (листинг 2.60).

Листинг 2.59. Содержимое файла Func.pm

```
package Func;
use Exporter;
our @ISA = qw( Exporter );
our @EXPORT = qw( &f_Func1 );
our @EXPORT_OK = qw ( &f_Func2 );
sub f_Func1 {
   return 5;
}
sub f_Func2 {
   return 7;
}
return 1; # означает, что файл успешно подключен
```

Листинг 2.60. Явное указание списка импорта

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use Func qw( :DEFAULT &f_Func2 );
print &f_Func1(), "<BR>";
print &f_Func2();
```

Обратите внимание на строку:

```
qw( :DEFAULT &f_Func2 );
```

Здесь мы явным образом импортируем название функции `f_Func2()`. Спецификация `:DEFAULT` включает в список импорта все идентификаторы из массива `@EXPORT`. Если не сделать так, то к функции `f_Func1()` можно обратиться, только предварительно указав название модуля:

```
print &Func::f_Func1(), "<BR>";
```

Иначе выражение:

```
print &f_Func1(), "<BR>";
```

вызовет ошибку. В нашем случае вместо спецификации `:DEFAULT` можно просто указать идентификатор функции `f_Func1()`:

```
use Func qw( &f_Func1 &f_Func2 );
```

В предыдущих примерах мы импортировали из модуля всего две функции. Но количество функций, требующих импорта, может исчисляться десятками и даже сотнями. Перечисление всех функций в списке импорта может стать утомительным занятием. Ассоциативный массив `%EXPORT_TAGS` позволяет объединить функции в группы и назначить этим группам имена:

```
our %EXPORT_TAGS = (
   "tag1" => [qw( &f_Func1 &f_Func2 )],
   "tag2" => [qw( &f_Func3 &f_Func4 )]
);
```

В этом примере мы создали две группы — `tag1` и `tag2`. Группа `tag1` объединяет функции `f_Func1()` и `f_Func2()`, а группа `tag2` — функции `f_Func3()` и `f_Func4()`. Добавить группу имен в массив `@EXPORT` можно следующим образом:

```
Exporter::export_tags('tag1');
```

Для добавления группы в массив @EXPORT_OK используется следующий синтаксис:

```
Exporter::export_ok_tags('tag2');
```

Теперь вместо перечисления идентификаторов в списке импорта достаточно указать имя группы. Перед именем группы следует добавить двоеточие:

```
use Func qw( :tag1 :tag2 );
```

2.13. Объектно-ориентированное программирование

Класс — это тип объекта, включающий множество переменных и функций для управления этими переменными. Переменные называют *свойствами*, а функции — *методами*. В языке Perl класс реализуется в виде модуля.

Для использования методов и свойств класса необходимо создать экземпляр класса. Для этого обычно используется метод new. После названия метода указывается имя класса, к которому будет относиться данный экземпляр. После имени класса, в круглых скобках, можно передавать некоторые параметры, задавая таким образом начальные значения свойствам класса:

```
<Экземпляр класса> = new <Имя класса> ([<Параметры>]);
```

При обращении к методам используется следующий формат:

```
<Экземпляр класса>-><Имя метода>([<Параметры>]);
```

Создать экземпляр класса можно, используя формат обращения к методам класса:

```
<Экземпляр класса> = <Имя класса>->new([<Параметры>]);
```

Метод, возвращающий экземпляр класса (ссылку на объект), не обязательно должен называться new. Мы можем использовать абсолютно любое допустимое название. Например, название add:

```
<Экземпляр класса> = <Имя класса>->add([<Параметры>]);
```

2.13.1. Создание класса

Рассмотрим создание класса на примере. Создадим класс с названием MyClass. Для этого в папке с исполняемым файлом создаем модуль с названием MyClass.pm (листинг 2.61).

Листинг 2.61. Содержимое файла MyClass.pm

```
package MyClass;
use strict;

my $str = "Строка"; # Закрытая переменная
our $public_str = "Открытая переменная";
sub new { # Конструктор
   my $invocant = shift();
   my $class = ref($invocant) || $invocant;
   my $self = { "var" => 10, @_ };
   print $self, "<BR>";
   print ref($self), "<BR>";
   bless($self, $class);
   return $self;
}
sub f_display { # Выводим значения переменных
   my $class = shift();
   print "Переменная var = ", $class->{"var"}, "<BR>";
   print "Переменная \$str = ", $class->f_get() , "<BR>";
}
sub f_set { # Присваиваем новое значение переменной $str
   my $class = shift();
   $str = shift();
}
sub f_get { # Возвращаем значение переменной $str
   return $str;
}
sub AUTOLOAD { # Если метода нет
   my $name = our $AUTOLOAD;
   print "Вызван неизвестный метод $name<BR>";
}
sub DESTROY { # Деструктор
   print "Вызван метод DESTROY<BR>";
}
1;
```

Теперь создадим исполняемый файл (листинг 2.62).

Листинг 2.62. Исполняемый файл

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";
```

```
use MyClass;
my $obj = MyClass->new("var" => 20);
print $obj, "<BR>";
print ref($obj), "<BR><BR>";

print "<B>Вывод переменных напрямую:</B><BR>";
print "Переменная \$public_str = ", $MyClass::public_str, "<BR>";
print "Переменная \$str = ", $MyClass::str;
print ", не доступна напрямую<BR>";

print "<B>Вывод переменных до изменения:</B><BR>";
print "Переменная \$str = ", $obj->f_get() , "<BR>";
$obj->f_display();

$obj->f_set("Новая строка");
$obj->{"var"} = 8;

print "<B>Вывод переменных после изменения:</B><BR>";
$obj->f_display();
print "<BR>";
$obj->f_print();
```

Вывод:

```
HASH(0x225fd8)
HASH
MyClass=HASH(0x225fd8)
MyClass

Вывод переменных напрямую:
Переменная $public_str = Открытая переменная
Переменная $str = , не доступна напрямую
Вывод переменных до изменения:
Переменная $str = Строка
Переменная var = 20
Переменная $str = Строка
Вывод переменных после изменения:
Переменная var = 8
Переменная $str = Новая строка

Вызван неизвестный метод MyClass::f_print
Вызван метод DESTROY
```

Итак, сначала мы подключаем модуль с классом посредством оператора `use`:

```
use MyClass;
```

В следующей строке создаем экземпляр класса `MyClass` с помощью метода `new()`. Метод, возвращающий экземпляр класса, называется *конструктором*:

```
my $obj = MyClass->new("var" => 20);
```

Данное выражение эквивалентно вызову метода `new()` следующим образом:

```
my $obj = MyClass::new("MyClass", "var" => 20);
```

Таким образом, первым элементом массива `@_` будет название создаваемого класса. Далее мы сохраняем имя класса в переменной `$invocant` с помощью выражения:

```
my $invocant = shift();
```

Так как в ряде случаев вместо имени класса может быть объект, то с помощью функции `ref()` мы определяем тип объекта. Если тип объекта не определен, то используется имя класса:

```
my $class = ref($invocant) || $invocant;
```

Одновременно с созданием экземпляра класса мы присваиваем начальное значение переменной `var`. Для хранения переменных класса используем анонимный хеш, ссылку на который сохраняем в переменной `$self`:

```
my $self = { "var" => 10, @_ };
```

Применение анонимного хеша не является обязательным. Вместо него можно использовать анонимный массив. Доступ к массиву осуществляется быстрее, чем к хешу, но на практике более удобно использовать именно хеш.

В начале анонимного хеша мы присваиваем переменной `var` значение по умолчанию. Создадим экземпляр класса следующим образом:

```
my $obj = MyClass->new();
```

В этом случае переменная `var` будет иметь значение 10. Но т. к. во втором значении анонимного хеша мы указали массив `@_` и передали новое значение конструктору класса, то значение по умолчанию будет замещено новым значением. В нашем случае переменная `var` будет иметь значение 20, а не 10.

С помощью функции `bless()` мы преобразуем ссылку на анонимный хеш в ссылку на объект, а затем возвращаем эту ссылку:

```
bless($self, $class);
return $self;
```

До применения функции `bless()` переменная `$self` содержала адрес анонимного хеша:

```
HASH(0x225f18)
```

А функция `ref()` для этой ссылки возвращала значение "HASH". После применения функции `bless()` и присвоения ссылки переменная `$obj` содержит следующее значение:

```
MyClass=HASH(0x225f18)
```

Функция `ref()` для этой ссылки возвращает уже имя класса — `MyClass`. Таким образом, функция `bless()` как бы "благословила" ссылку на объект.

Для демонстрации применения закрытых и открытых переменных мы создали внутри модуля две переменные — `$public_str` и `$str`:

```
my $str = "Строка";
our $public_str = "Открытая переменная";
```

Переменная `$public_str` объявлена как глобальная переменная пакета. По этой причине мы можем получить или изменить значение переменной, используя синтаксис `$MyClass::public_str`:

```
print "Переменная \$public_str = ", $MyClass::public_str, "<BR>";
```

В некоторых случаях требуется контролировать значение переменной. Например, если в переменной предполагается хранение E-mail-адреса, то перед присвоением значения мы можем с помощью регулярных выражений проверить соответствие значения некоторому шаблону. Если же любой пользователь будет иметь возможность ввести что угодно, минуя нашу проверку, то ни о каком контроле не может быть и речи. Эмулировать закрытую переменную можно с помощью лексической переменной. Лексические переменные не помещаются в таблицу имен пакета, а это значит, что получить или изменить значение переменной с помощью синтаксиса `$MyClass::str` уже не получится:

```
print "Переменная \$str = ", $MyClass::str;
print ", т. е. недоступна напрямую<BR>";
```

Для присвоения значения закрытой переменной `$str` используется метод `f_set()`:

```
sub f_set {
   my $class = shift();
   $str = shift();
}
```

Именно внутрь этого метода мы можем вставить код проверки вводимого значения. В программе метод вызывается следующим образом:

```
$obj->f_set("Новая строка");
```

Чтобы получить текущее значение закрытой переменной, мы определили в классе метод `f_get()`:

```
sub f_get {
   return $str;
}
```

В программе метод `f_get()` вызывается следующим образом:

```
print "Переменная \$str = ", $obj->f_get(), "<BR>";
```

Для получения или изменения значения переменной внутри хеша используется следующий синтаксис:

```
print "Переменная var = ", $obj->{"var"};
$obj->{"var"} = 8;
```

Кроме того, для вывода значений закрытой переменной и переменной внутри хеша мы создали метод `f_display()`:

```
sub f_display {
   my $class = shift();
   print "Переменная var = ", $class->{"var"}, "<BR>";
   print "Переменная \$str = ", $class->f_get(), "<BR>";
}
```

Метод `f_display()` является открытым методом. В некоторых случаях необходимо закрыть общий доступ к определенному методу. Закрытые методы реализуются через создание ссылки на анонимную функцию. Причем для хранения ссылки должна быть использована лексическая переменная:

```
my $link_sub = sub {
# "Тело" функции
}
```

Если запрашиваемый метод не определен внутри модуля, то интерпретатор вызывает предопределенный метод `AUTOLOAD()`. Если внутри модуля создать этот метод, то с помощью глобальной переменной `$AUTOLOAD` мы можем получить имя неопределенного метода и обработать исключительную ситуацию:

```
sub AUTOLOAD {
   my $name = our $AUTOLOAD;
   print "Вызван неизвестный метод $name<BR>";
}
```

Для примера мы вызываем несуществующий метод `f_print()`:

```
$obj->f_print();
```

В итоге получим следующее сообщение:

```
Вызван неизвестный метод MyClass::f_print
```

Если в модуле нет метода AUTOLOAD, то получим следующее сообщение об ошибке:

```
Can't locate object method "f_print" via package "MyClass"
```

Как вы уже знаете, метод, создающий и возвращающий ссылку на объект, называется конструктором. Если конструктор вызывается при создании объекта, то перед уничтожением объекта автоматически вызывается метод, называемый *деструктором*. В языке Perl деструктор реализуется в виде предопределенного метода DESTROY(). В качестве параметра методу DESTROY() передается ссылка на уничтожаемый объект:

```
sub DESTROY {
   print "Вызван метод DESTROY<BR>";
}
```

Так как управление памятью в языке Perl производится самим интерпретатором, то деструктор требуется достаточно редко.

2.13.2. Наследование

Предположим, у нас есть класс (например, Class1). При помощи *наследования* мы можем создать новый класс (например, Class2), в котором будет доступ ко всем методам класса Class1 плюс некоторые новые методы.

Для примера в одной папке создадим три файла:

- Class1.pm (листинг 2.63) — для реализации класса Class1;
- Class2.pm (листинг 2.64) — для реализации класса Class2;
- test.pl (листинг 2.65) — основная программа.

Листинг 2.63. Содержимое файла Class1.pm

```
package Class1;
use strict;

sub new { # Конструктор
   my $invocant = shift();
   my $class = ref($invocant) || $invocant;
   my $self = {};
   bless($self, $class);
   return $self;
}
sub f_print {
   print "Метод f_print класса Class1<BR>";
}
```

```
sub f_display {
   print "Метод f_display класса Class1<BR>";
}
1;
```

Листинг 2.64. Содержимое файла Class2.pm

```
package Class2;
use strict;

use Class1;
our @ISA = ("Class1");
sub new { # Конструктор
   my $invocant = shift();
   my $class = ref($invocant) || $invocant;
   my $self = $class->SUPER::new();
   bless($self, $class);
   return $self;
}
sub f_new_method() {
   print "Метод f_new_method класса Class2<BR>";
}
sub f_display {
   print "Метод f_display класса Class2<BR>";
}
1;
```

Листинг 2.65. Код основной программы

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

use Class2;
my $obj = Class2->new();
$obj->f_new_method();
$obj->f_print();
$obj->f_display();

print "<BR>";
if ($obj->isa("Class1")) {
   print "\$obj принадлежит классу Class1<BR>";
}
```

```
if ($obj->isa("HASH")) {
   print "\$obj ссылка на HASH<BR>";
}
if ($obj->can("f_new_method")) {
   print "Метод f_new_method() определен<BR>";
}
else {
   print "Метод f_new_method() не определен<BR>";
}
if ($obj->can("f_new")) {
   print "Метод f_new() определен<BR>";
}
else {
   print "Метод f_new() не определен<BR>";
}
```

Итак, в классе Class1 определены три метода: `new()` (конструктор класса), `f_print()` и `f_display()`. В классе Class2 также определены три метода — `new()` (конструктор класса), `f_new_method()` и `f_display()`.

Для наследования классом Class2 всех методов класса Class1 необходимо указать в глобальном массиве `@ISA` класс Class1:

```
use Class1;
our @ISA = ("Class1");
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> При наследовании не производится автоматическое подключение модуля с наследуемым классом. По этой причине необходимо предварительно подключить модуль с классом с помощью оператора `use`.

В основной программе мы подключаем модуль только с классом Class2. Затем создаем экземпляр класса:

```
use Class2;
my $obj = Class2->new();
```

Следует обратить внимание на следующую строку кода конструктора класса Class2:

```
my $self = $class->SUPER::new();
```

С помощью псевдокласса `SUPER` мы наследуем конструктор от класса Class1. Используя псевдокласс `SUPER`, можно вызвать метод базового класса из текущего класса. При этом можно добавить до или после вызова какие-то собственные действия.

Если указанный метод найден в текущем классе, то он будет выполнен:

```
$obj->f_new_method();
```

Вывод:

```
Метод f_new_method класса Class2
```

Если метода нет в текущем классе, то производится поиск в базовых классах, указанных в глобальном массиве `@ISA`:

```
$obj->f_print();
```

Вывод:

```
Метод f_print класса Class1
```

Если имя метода в классе Class2 совпадает с именем метода класса Class1, то будет использоваться метод из класса Class2:

```
$obj->f_display();
```

Вывод:

```
Метод f_display класса Class2
```

Если, конечно, мы не переопределим метод с помощью псевдокласса `SUPER`:

```
sub f_display {
   my $self = shift();
   $self->SUPER::f_display();
}
```

Вывод:

```
Метод f_display класса Class1
```

При множественном наследовании все наследуемые классы перечисляются в глобальном массиве `@ISA`. При этом поиск неопределенного в текущем классе метода производится в указанных в списке базовых классах слева направо. Если метод не найден в базовых классах, то поиск метода производится в особом предопределенном классе `UNIVERSAL`. Все классы неявно наследуют следующие методы класса `UNIVERSAL`:

- `isa(<Имя класса или встроенный тип>)` — возвращает `true`, если указанный в качестве параметра класс принадлежит текущему классу или является наследником текущего класса:

```
use Class2;
my $obj = Class2->new();
if ($obj->isa("Class1")) {
   print "\$obj принадлежит классу Class1<BR>";
}
```

Вывод:

```
$obj принадлежит классу Class1
```

Если в качестве параметра указать встроенный тип (например, ARRAY или HASH), то можно определить точный тип ссылки:

```
use Class2;
my $obj = Class2->new();
if ($obj->isa("HASH")) {
   print "\$obj ссылка на HASH<BR>";
}
```

Вывод:

```
$obj ссылка на HASH
```

- can(<Метод>) — возвращает undef, если указанный в качестве параметра метод не найден в текущем или базовых классах. Позволяет проверить наличие метода перед его вызовом:

```
use Class2;
my $obj = Class2->new();
if ($obj->can("f_new_method")) {
   print "Метод f_new_method() определен<BR>";
}
else {
   print "Метод f_new_method() не определен<BR>";
}
if ($obj->can("f_new")) {
   print "Метод f_new() определен<BR>";
}
else {
   print "Метод f_new() не определен<BR>";
}
```

Вывод:

```
Метод f_new_method() определен
Метод f_new() не определен
```

2.13.3. Создание шаблона сайта при помощи класса

При создании больших сайтов обычно страницу делят на три части: верхний колонтитул, тело страницы и нижний колонтитул. Для подключения колонтитулов к основному документу используются операторы require и use.

Нижний колонтитул практически всегда одинаков для всех страниц, а вот верхний изначально не может быть одинаковым. Для всех страниц сайта нельзя использовать один и тот же заголовок (тег <TITLE>). Более того, каждая страница должна иметь уникальное описание для поисковых роботов (тег <META>).

Для реализации верхнего колонтитула создадим класс, позволяющий менять заголовок и описание страницы. Для примера создадим два файла: Header.pm (листинг 2.66) — верхний колонтитул, index.pl (листинг 2.67) — основное содержание страницы.

Листинг 2.66. Содержимое файла Header.pm

```
package Header;
use strict;

sub new {
   my $invocant = shift();
   my $class = ref($invocant) || $invocant;
   my $self = {
      "Title" => "",
      "Description" => "",
      @_
   };
   bless($self, $class);
   return $self;
}
sub f_display() {
   my $self = shift();
   print <<HTML_COD;
<HTML><HEAD>
<TITLE>$self->{"Title"}</TITLE>
<META name="description" content="$self->{"Description"}">
<META http-equiv="Content-Type" content="text/html; charset=windows-1251">
</HEAD>
<BODY bgcolor="#FFFFFF">
HTML_COD
}
1;
```

Листинг 2.67. Содержимое файла index.pl

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
```

```
use strict;
print "Content-type: text/html\n\n";

use Header;
my $obj = Header->new(
   "Title" => "Заголовок",
   "Description" => "Описание"
);
$obj->f_display();
print "Основное содержание страницы\n";
print "</BODY>\n";
print "</HTML>\n";
```

Если открыть файл index.pl в Web-браузере и отобразить исходный код, то мы получим:

```
<HTML><HEAD>
<TITLE>Заголовок</TITLE>
<META name="description" content="Описание">
<META http-equiv="Content-Type" content="text/html; charset=windows-
1251">
</HEAD>
<BODY bgcolor="#FFFFFF">
Основное содержание страницы
</BODY>
</HTML>
```

Таким образом можно сделать уникальными заголовок и описание каждой страницы.

2.14. Ошибки в программе

Существуют три типа ошибок в скриптах: синтаксические, логические и ошибки времени выполнения.

2.14.1. Синтаксические ошибки

Синтаксические ошибки — это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей скобок и т. д. Это ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки. А программа не будет выполняться совсем.

Например, если вместо

```
print "<BR>";
```

написать

```
prinnt "<BR>";
```

В Web-браузере отобразится нечто подобное:

```
syntax error at Z:/home/perlbook.ru/cgi-bin/index.pl line 7, near "prinnt
"<BR>""
Execution of Z:/home/perlbook.ru/cgi-bin/index.pl aborted due to
compilation errors.
```

Итак, интерпретатор предупреждает нас, что в строке 7 файла index.pl содержится ошибка. Достаточно отсчитать 7 строк в исходном коде и исправить опечатку с `prinnt` на `print`.

Перечислим часто встречающиеся синтаксические ошибки:

- ❒ отсутствует точка с запятой в конце выражения;
- ❒ опечатка в имени оператора или функции;
- ❒ буква набрана в русской раскладке клавиатуры, вместо латинской;
- ❒ отсутствие открывающей или закрывающей скобки (или наоборот лишние скобки);
- ❒ в логическом выражении вместо оператора == (равно), указан оператор присваивания =;
- ❒ в логическом выражении вместо операторов сравнения строк указаны операторы сравнения чисел, и наоборот;
- ❒ в цикле `for` указаны параметры через запятую, а не через точку с запятой.

2.14.2. Логические ошибки

Логические ошибки — это ошибки в логике работы программы, которые можно выявить только по результатам работы скрипта. Как правило, интерпретатор не предупреждает о наличии ошибки. А программа будет выполняться, т. к. не содержит синтаксических ошибок. Такие ошибки достаточно трудно выявить и исправить.

2.14.3. Ошибки времени выполнения

Ошибки времени выполнения — это ошибки, которые возникают во время работы скрипта. Причиной являются события, не предусмотренные программистом. Классическим примером служит деление на ноль:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
```

```
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

my $var = 20;
for (my $i=10; $i>=0; $i--) {
   print ($var/$i);
   print "<BR>";
}
```

Через некоторое время переменная `$i` получит значение 0. В результате получим ошибку времени выполнения:

```
Illegal division by zero at Z:/home/perlbook.ru/cgi-bin/index.pl line 9.
```

2.14.4. Вывод сообщений об ошибках

Для вывода предупреждающих сообщений при компиляции программы необходимо после пути к интерпретатору указать флаг `-w`:

```
#!/usr/bin/perl -w
```

Все сообщения об ошибках обычно записываются в журнал ошибок (error.log). Логи сервера можно найти с папке C:\WebServers\usr\local\apache \logs. Просмотреть содержимое журнала позволяет любой текстовый редактор, например Блокнот. Но каждый раз просматривать этот журнал не очень удобно. Для вывода сообщений об ошибках в окно Web-браузера мы указываем в программе следующую строку:

```
use CGI::Carp qw(fatalsToBrowser);
```

С помощью оператора `use` мы подключаем модуль `CGI::Carp`. Теперь при возникновении ошибки можно сразу увидеть ее описание.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В описании ошибки обычно указывается номер строки, в которой содержится ошибка. Для ее исправления достаточно отсчитать указанное количество строк и исправить ошибку. Если указана последняя строка в программе, то, скорее всего, программист забыл указать закрывающую скобку в любом месте программы.

Выводить сообщения об ошибках в окно Web-браузера стоит только на этапе разработки программы. После загрузки программы на сервер следует отключить это "удобство": чем меньше пользователь получает сведений об ошибках, тем лучше. Ведь злоумышленник может использовать эту информацию против вас. Однако администратор должен знать об ошибках. Функция `carpout()` позволяет выводить сообщения не в стандартный журнал ошибок, а в указанный файл. Для этого функции следует передать ссылку на дескрип-

тор ранее открытого файла. Чтобы получить сообщения об ошибках на этапе компиляции, необходимо разместить функцию в блоке BEGIN:

```
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
```

2.14.5. Инструкция *or die()*

Для обработки ошибок можно использовать инструкцию or die(). В круглых скобках указывается сообщение об ошибке:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

open(FILE, 'file.txt') or die("Ошибка при открытии файла. $!");
```

При отсутствии файла file.txt будет выведено сообщение:

```
Ошибка при открытии файла. No such file or directory at
Z:/home/perlbook.ru/cgi-bin/index.pl line 7.
```

Дальнейшее выполнение программы будет остановлено. В специальной переменной $! сохраняется подробное описание ошибки:

```
No such file or directory
```

2.14.6. Прагмы *strict* и *warnings*

Чтобы избежать множества ошибок и выявить их на этапе компиляции скрипта, следует воспользоваться прагмами strict и warnings. Подключить прагмы позволяет оператор use.

Прагма strict заставляет программиста явно объявлять переменные и функции. Рассмотрим следующий пример:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

$var = 10;
print $vat;
```

В этом примере мы намеренно сделали ошибку в имени скалярной переменной. В итоге, в программе якобы используются две переменные, и компилятор не выдает никаких ошибок. Если в программу включить прагму `strict`, то результат будет совсем другим:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use strict;
$var = 10;
print $vat;
```

В итоге получим следующие сообщения об ошибках:

```
Global symbol "$var" requires explicit package name at
Z:/home/perlbook.ru/cgi-bin/index.pl line 8.
Global symbol "$vat" requires explicit package name at
Z:/home/perlbook.ru/cgi-bin/index.pl line 9.
Execution of Z:/home/perlbook.ru/cgi-bin/index.pl aborted due to
compilation errors.
```

Теперь исправим опечатку в имени переменной:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use strict;
$var = 10;
print $var;
```

В итоге опять получим ошибки:

```
Global symbol "$var" requires explicit package name at
Z:/home/perlbook.ru/cgi-bin/index.pl line 8.
Global symbol "$var" requires explicit package name at
Z:/home/perlbook.ru/cgi-bin/index.pl line 9.
Execution of Z:/home/perlbook.ru/cgi-bin/index.pl aborted due to
compilation errors.
```

И так будет до тех пор, пока мы явно не объявим переменную с помощью ключевого слова `our` или `my`:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use strict;
my $var = 10;
print $var;
# Выведет 10
```

Прагме `strict` можно указать, что проверять, переменные или функции. Для этого необходимо передать следующие аргументы:

- `vars` — для проверки переменных:
  ```
  use strict 'vars';
  ```
- `subs` — для проверки функций:
  ```
  use strict 'subs';
  ```

Применение прагмы `warnings` аналогично использованию флага `-w` после пути к интерпретатору. Все сообщения записываются в журнал ошибок сервера Apache (C:\WebServers\usr\local\apache\logs\error.log):

```
use warnings;
```

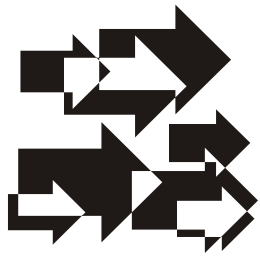
Аналогично:

```
#!/usr/bin/perl -w
```

Отключить прагмы позволяет оператор `no`.

```
no strict;
no warnings;
```

ГЛАВА 3



Web-программирование на Perl

3.1. Переменные окружения

Создадим сценарий, состоящий всего из шести строк:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
$var=10;
```

А теперь вопрос. Сколько переменных доступно сценарию? Думаете, одна `$var`? Давайте перепишем нашу программу и добавим одну строку:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";
$var=10;
print $ENV{'DOCUMENT_ROOT'};
```

В результате работы скрипта в окне Web-браузера отобразится следующая строка:

```
Z:/home/perlbook.ru/www
```

Откуда же взялась переменная `$ENV{'DOCUMENT_ROOT'}`? Ведь мы ее не создавали! Ответ на этот вопрос достаточно прост: эта переменная была автоматически создана сервером. Такая переменная называется *переменной окружения*.

3.1.1. Ассоциативный массив *%ENV*

Все переменные окружения доступны через ассоциативный массив `%ENV`. Выведем все переменные окружения, которые доступны сценарию:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

my $var;
foreach $var (keys(%ENV)) {
   print "$var => $ENV{$var}\n";
}
```

Результат выполнения скрипта в исходном HTML-коде приведен в листинге 3.1.

Листинг 3.1. Ассоциативный массив `%ENV`

```
SCRIPT_NAME => /cgi-bin/index.pl
SERVER_NAME => perlbook.ru
SERVER_ADMIN => admin@localhost
HTTP_ACCEPT_ENCODING => deflate, gzip, x-gzip, identity, *;q=0
HTTP_CONNECTION => Keep-Alive, TE
REQUEST_METHOD => GET
SYSTEMROOT => C:\WINDOWS
HTTP_ACCEPT => text/html, application/xml;q=0.9, application/xhtml+xml,
image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
SCRIPT_FILENAME => Z:/home/perlbook.ru/cgi-bin/index.pl
COMSPEC => C:\WINDOWS\system32\cmd.exe
SERVER_SOFTWARE => Apache/2.2.4 (Win32) mod_ssl/2.2.4 OpenSSL/0.9.8d
PHP/5.2.4
HTTP_ACCEPT_CHARSET => iso-8859-1, utf-8, utf-16, *;q=0.1
WINDIR => C:\WINDOWS
HTTP_TE => deflate, gzip, chunked, identity, trailers
QUERY_STRING =>
REMOTE_PORT => 1129
PATHEXT => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
HTTP_USER_AGENT => Opera/9.02 (Windows NT 5.1; U; ru)
```

```
SERVER_PORT => 80
SERVER_SIGNATURE => <address>Apache/2.2.4 (Win32) mod_ssl/2.2.4
OpenSSL/0.9.8d PHP/5.2.4 Server at perlbook.ru Port 80</address>


HTTP_CACHE_CONTROL => no-cache
HTTP_ACCEPT_LANGUAGE => ru-RU,ru;q=0.9,en;q=0.8
REMOTE_ADDR => 127.0.0.1
SERVER_PROTOCOL => HTTP/1.1
PATH =>
\usr\local\ImageMagick;\usr\local\php5;C:\WINDOWS\system32;C:\WINDOWS;C:\
WINDOWS\System32\Wbem;C:\Program
Files\Intel\DMIX;C:\WebServers\usr\local\perl\bin
REQUEST_URI => /cgi-bin/index.pl
GATEWAY_INTERFACE => CGI/1.1
SERVER_ADDR => 127.0.0.1
DOCUMENT_ROOT => Z:/home/perlbook.ru/www
HTTP_HOST => perlbook.ru
```

3.1.2. Часто используемые переменные окружения

Перечислим наиболее часто используемые переменные окружения:

- `$ENV{'DOCUMENT_ROOT'}` — путь к корневому каталогу сервера;
- `$ENV{'REMOTE_ADDR'}` — IP-адрес запрашивающего ресурс клиента;
- `$ENV{'REMOTE_USER'}` — имя пользователя, прошедшего аутентификацию;
- `$ENV{'QUERY_STRING'}` — строка переданных серверу параметров;
- `$ENV{'HTTP_USER_AGENT'}` — название и версия Web-браузера клиента;
- `$ENV{'HTTP_REFERER'}` — URL-адрес, с которого пользователь перешел на наш сайт;
- `$ENV{'REQUEST_METHOD'}` — метод передачи информации (`GET` или `POST`).

Остальные переменные окружения используются реже, а по названиям интуитивно понятно их предназначение. В дальнейшем мы еще не раз будем возвращаться к переменным окружения.

3.2. Заголовки HTTP

Заголовки HTTP предназначены для передачи некоторых дополнительных сведений, например: при запросе файла Web-браузером дополнительно указываются предпочитаемые MIME-типы, поддерживаемые языки и кодировки,

информация о самом Web-браузере и др. Сервер, в свою очередь, при выдаче файла указывает MIME-тип файла, дату последней модификации файла, сведения о кодировке, языке и т. д.

Перечислим основные заголовки HTTP:

- `Accept` — содержит MIME-типы, поддерживаемые Web-браузером:

```
Accept: text/html, application/xml;q=0.9, application/xhtml+xml,
image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
```

- `Accept-Language` — список поддерживаемых Web-браузером языков в порядке предпочтения:

```
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
```

- `Accept-Charset` — список поддерживаемых Web-браузером кодировок:

```
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
```

- `Accept-Encoding` — список поддерживаемых Web-браузером методов сжатия:

```
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
```

- `Content-Type` — предназначен для указания типа передаваемых данных:

```
Content-Type: text/plain; charset=windows-1251
```

- `Content-Length` — длина передаваемых данных при методе `POST`:

```
Content-Length: 0
```

- `Cookie` — содержит информацию об установленных cookies:
- `Set-Cookie` — позволяет установить cookies:

```
Set-Cookie: id=85; domain=.perlbook.ru; path=/; expires=Sun,
22-Jun-2008 16:57:07 GMT
```

- `GET` — заголовок при передаче данных методом `GET`;
- `POST` — заголовок при передаче данных методом `POST`;
- `Last-Modified` — содержит дату последней модификации файла:

```
Last-Modified: Sun, 27 May 2007 21:58:21 GMT
```

- `Location` — при указании этого заголовка Web-браузер обязан перейти по указанному URL-адресу:

```
Location: firma.pl
```

- `Pragma` — используется для запрета кэширования документа:

```
Pragma: no-cache
```

- `Referer` — содержит URL-адрес, с которого перешел пользователь на наш сайт;
- `Server` — содержит название и версию программного обеспечения сервера:

```
Server: Apache/2.0.59 (Win32) Perl/5.2.2
```

- `User-Agent` — содержит информацию об используемом Web-браузере:

```
User-Agent: Opera/9.02 (Windows NT 5.1; U; ru)
```

3.2.1. Перенаправление клиента на другой URL-адрес. Запрет кэширования страниц

Чтобы перенаправить клиента на URL **http://www.rambler.ru/**, нужно написать следующий код:

```
#!/usr/bin/perl -w
print "Location: http://www.rambler.ru/\n\n";
```

Пример запрета кэширования страниц приведен в листинге 3.2.

Листинг 3.2. Запрет кэширования страниц

```
#!/usr/bin/perl -w
print "Expires: Sun, 27 May 2008 01:00:00 GMT\n";
print "Last-Modified: " . &f_gmdate() . " GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html\n\n";

# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);

print "Текст файла<BR>";

sub f_gmdate {
   my @date = gmtime();
   my @day = ("Sun", "Mon", "Tue", "Wed", "Thu",
        "Fri", "Sat");
   my @month = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
         "Aug", "Sep", "Oct", "Nov", "Dec");
```

```
    my $year = $date[5] + 1900;
    my $gmdate = "";
    $gmdate = $day[$date[6]];
    if (length($date[3]) == 1) {
       $date[3] = "0" . $date[3];
    }
    $gmdate .= ", " . $date[3] . " ";
    $gmdate .= $month[$date[4]];
    $gmdate .= " " . $year . " ";
    if (length($date[2]) == 1) {
       $date[2] = "0" . $date[2];
    }
    if (length($date[1]) == 1) {
       $date[1] = "0" . $date[1];
    }
    if (length($date[0]) == 1) {
       $date[0] = "0" . $date[0];
    }
    $gmdate .= $date[2] . ":" . $date[1] . ":" . $date[0];
    return $gmdate;
}
```

3.2.2. Работа с cookies. Создаем индивидуальный счетчик посещений

Web-браузеры позволяют сохранять небольшой объем информации в специальном текстовом файле на компьютере пользователя — cookies. Возможность использования cookies можно отключить в настройках Web-браузера.

Для записи cookies используется заголовок `Set-Cookie`. Заголовок имеет следующий формат:

```
Set-Cookie: <Имя>=<Значение>; [expires=<Время жизни>]; [path=<Путь>];
[domain=<Домен>]
```

Например:

```
Set-Cookie: id=85; domain=.perlbook.ru; path=/; expires=Sun, 22-Jun-2008
16:57:07 GMT
```

В этом примере мы устанавливаем cookies с именем `id` и значением 85 сроком до 22 июня 2008 г. для домена perlbook.ru. Cookies будет доступен всем документам домена.

При установке cookies следует учитывать, что имя и значение не должны иметь пробелов, табуляции или точки с запятой. Чтобы сохранить в cookies большой объем информации, необходимо применить URL-кодирование. Это полезно также при сохранении в cookies символов русского алфавита.

Большинство параметров не являются обязательными. Если не указано время жизни cookies, они удаляются сразу после закрытия Web-браузера.

Сформировать строку с параметрами позволяет функция `cookie()` из модуля `CGI`. Функция имеет следующий формат:

```
cookie(-name=>'<Имя>', -value=>'<Значение>', [-expires=>'<Время жизни>'],
[-path=>'<Путь>'], [-domain=>'<Домен>'])
```

В параметре `<Время жизни>` можно указать срок в днях или часах. Например:

```
+10d
+3h
```

В первом случае cookies установлен на 10 дней, а во втором на 3 часа:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
my $cookies1 = cookie(-name=>'id1', -value=>'85', -expires=>'+10d',
-path=>'/', -domain=>'.perlbook.ru');
my $cookies2 = cookie(-name=>'id2', -value=>'Пример использования',
-expires=>'+3h', -path=>'/', -domain=>'.perlbook.ru');
print "Content-type: text/html\n\n";

print $cookies1, "<BR>";
print $cookies2;
```

Вывод:

```
id1=85; domain=.perlbook.ru; path=/; expires=Sun, 22-Jun-2008 17:28:49 GMT
id2=%CF%F0%E8%EC%E5%F0%20%E8%F1%EF%EE%EB%FC%E7%EE%E2%E0%ED%E8%FF;
domain=.perlbook.ru; path=/; expires=Thu, 12-Jun-2008 20:28:49 GMT
```

Как видите, функция `cookie` не только избавляет нас от вычисления даты, но и производит URL-кодирование значения.

Все установленные cookies доступны через переменную окружения `$ENV{'HTTP_COOKIE'}`:

```
$cookies = $ENV{'HTTP_COOKIE'};
```

Переменная `$cookies` будет содержать строку, в которой перечислены все установленные пары `имя=значение` через точку с запятой:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
my $cookies1 = cookie(-name=>'id1', -value=>'85', -expires=>'+10d',
-path=>'/', -domain=>'.perlbook.ru');
my $cookies2 = cookie(-name=>'id2', -value=>'Пример использования',
-expires=>'+3h', -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies1\n";
print "Set-Cookie: $cookies2\n";
print "Content-type: text/html\n\n";

my $cookies = $ENV{'HTTP_COOKIE'};
print $cookies;
```

Вывод после перезагрузки страницы:

```
id1=85; id2=%CF%F0%E8%EC%E5%F0%20%E8%F1%EF%EE%EB%FC%E7%EE%E2%E0%ED%E8%FF
```

Для получения значений достаточно разобрать эту конструкцию на составляющие. Например, так:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
my $cookies1 = cookie(-name=>'id1', -value=>'85', -expires=>'+10d',
-path=>'/', -domain=>'.perlbook.ru');
my $cookies2 = cookie(-name=>'id2', -value=>'Пример использования',
-expires=>'+3h', -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies1\n";
print "Set-Cookie: $cookies2\n";
print "Content-type: text/html\n\n";

my @cookies = split(/; /, $ENV{'HTTP_COOKIE'});
my %MyCookies;
```

```
my ($name, $value);
foreach (@cookies) {
   ($name, $value) = split(/=/, $_);
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $MyCookies{$name} = $value;
}
print $MyCookies{'id1'}, "<BR>";
print $MyCookies{'id2'};
```

Вывод:

```
85
Пример использования
```

Функция `cookie()` из модуля `CGI` предоставляет более простой способ доступа к установленным cookies. Для этого функции достаточно передать имя cookies, и функция автоматически произведет URL-декодирование значения:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
my $cookies1 = cookie(-name=>'id1', -value=>'85', -expires=>'+10d',
-path=>'/', -domain=>'.perlbook.ru');
my $cookies2 = cookie(-name=>'id2', -value=>'Пример использования',
-expires=>'+3h', -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies1\n";
print "Set-Cookie: $cookies2\n";
print "Content-type: text/html\n\n";

print cookie('id1'), "<BR>";
print cookie('id2');
```

Вывод:

```
85
Пример использования
```

Для удаления следует установить cookies с прошедшей датой.

В качестве примера использования cookies создадим счетчик посещений (листинг 3.3).

Листинг 3.3. Счетчик посещений

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
my $id_count;
if (!cookie('id_count')) {
   $id_count = 0;
}
else {
   $id_count = cookie('id_count');
}
$id_count++;
my $cookies = cookie(-name=>'id_count', -value=>$id_count,
-expires=>'+360d', -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies\n";
print "Content-type: text/html\n\n";

print "Вы посетили ресурс $id_count раз";
```

Теперь при каждом запросе страницы значение счетчика будет увеличиваться.

3.3. Обработка данных формы

В этом разделе мы рассмотрим два способа обработки данных формы. Первый способ является традиционным и использует переменные окружения и стандартный ввод, а второй способ заключается в применении модуля `CGI.pm`.

Вспомним технологию внедрения формы в HTML-документ. Добавить форму в HTML-документ позволяет парный тег `<FORM>`. Тег имеет следующие параметры:

- `action` — задает URL-адрес программы обработки формы:

  ```
  <FORM action="http://perlbook.ru/cgi-bin/file.pl">
  ```

- `method` — определяет, как будут пересылаться данные от формы до Web-сервера. Может принимать два значения — `GET` и `POST`:

- GET — применяется, когда объем пересылаемых данных невелик. Данные формы пересылаются путем их добавления к URL-адресу после знака ? в формате:

```
<Имя параметра>=<Значение параметра>
```

Каждая пара параметр=значение отделяются друг от друга символом &. Например:

```
http://perlbook.ru/cgi-bin/file.pl?pole1=Login&pole2=Password
```

Все специальные символы, а также буквы, отличные от латинских (например, буквы русского языка), кодируются в формате %nn, а пробел заменяется знаком +. Например, фраза "каталог сайтов" будет выглядеть следующим образом:

```
%EA%E0%F2%E0%EB%EE%E3+%F1%E0%E9%F2%EE%E2
```

А если эта фраза является значением поля с именем pole1, то строка запроса будет такой:

```
http://perlbook.ru/cgi-bin/file.pl?pole1=%EA%E0%F2%E0%EB%EE%E3+
%F1%E0%E9%F2%EE%E2&pole2=Password
```

В теге <FORM> значение GET для параметра method задается так:

```
<FORM action="http://perlbook.ru/cgi-bin/file.pl" method="GET">
```

- POST — предназначен для пересылки данных большого объема, файлов и конфиденциальной информации (например, паролей):

```
<FORM action="http://perlbook.ru/cgi-bin/file.pl" method="POST">
```

- enctype — задает MIME-тип передаваемых данных. Может принимать два значения:
  - application/x-www-form-urlencoded — применяется по умолчанию:

```
<FORM action="http://perlbook.ru/cgi-bin/file.pl" method="POST"
enctype="application/x-www-form-urlencoded">
```

  - multipart/form-data — указывается при пересылке Web-серверу файлов:

```
<FORM action="http://perlbook.ru/cgi-bin/file.pl" method="POST"
enctype="multipart/form-data">
```

Теперь приступим к разбору данных формы.

3.3.1. Метод *GET*

Для начала рассмотрим декодирование данных, отправленных методом GET. В качестве примера создадим форму с двумя текстовыми полями (листинг 3.4).

Листинг 3.4. Содержимое файла index.pl

```
<HTML>
<HEAD>
<TITLE>Обработка данных формы</TITLE>
</HEAD>
<BODY>
<B>Обработка данных формы</B>
<BR>
<FORM action="http://perlbook.ru/cgi-bin/form.pl" method="GET">
<INPUT type="text" name="txt1" size="20"><BR>
<INPUT type="text" name="txt2" size="20"><BR>
<INPUT type="submit" value="Передать">
</FORM>
</BODY>
</HTML>
```

Сохраним файл под названием index.html в папке C:\WebServers\home \perlbook.ru\www.

Теперь напишем программу для декодирования данных этой формы. Открываем Блокнот и набираем код из листинга 3.5.

Листинг 3.5. Декодирование данных формы

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my @form_data = split(/&/, $ENV{'QUERY_STRING'});
our %MyForm;
my ($name, $value);
foreach (@form_data) {
   ($name, $value) = split(/=/, $_);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $MyForm{$name} = $value;
}
```

```
print $MyForm{'txt1'}, "<BR>";
print $MyForm{'txt2'};
```

Сохраняем файл под названием form.pl в папке C:\WebServers\home \perlbook.ru\cgi-bin. Теперь открываем Web-браузер и в адресной строке набираем:

```
http://perlbook.ru/
```

В итоге отобразится форма с двумя текстовыми полями и кнопкой **Передать**. В качестве примера в первое поле вводим `строка 1`, а во второе — `строка 2`. Нажимаем кнопку **Передать**. После обработки данных формы будут выведены две строки:

```
строка 1
строка 2
```

Обратите внимание на адресную строку. После знака вопроса указаны следующие данные:

```
txt1=%F1%F2%F0%EE%EA%E0+1&txt2=%F1%F2%F0%EE%EA%E0+2
```

Получить эту строку в программе позволяет переменная окружения `$ENV{'QUERY_STRING'}`. Далее мы разбиваем строку по разделителю `&` и сохраняем фрагменты в массиве `@form_data` с помощью выражения:

```
my @form_data = split(/&/, $ENV{'QUERY_STRING'});
```

В итоге в массиве будут два элемента:

```
txt1=%F1%F2%F0%EE%EA%E0+1
txt2=%F1%F2%F0%EE%EA%E0+2
```

Затем в цикле декодируем полученные строки и заполняем ассоциативный массив `%MyForm`. В качестве ключа используется название поля, а в качестве значения данные этого поля:

```
$MyForm{$name} = $value;
```

Теперь остановимся на декодировании данных. В первой строке мы разбиваем на составляющие цепочку `параметр=значение` с помощью функции `split()` и присваиваем переменной `$name` имя поля, а переменной `$value` данные этого поля:

```
($name, $value) = split(/=/, $_);
```

Далее заменяем все знаки `+` на пробел:

```
$value =~ tr/+/ /;
```

Все специальные символы, а также буквы, отличные от латинских, закодированы в формате `%nn` с помощью выражения:

```
$value =~ s/%(..)/pack("C", hex($1))/eg;
```

Мы заменяем все последовательности %nn на соответствующие символы. Вначале находим эту последовательность с помощью выражения:

```
%(..)
```

Символ точка обозначает любой символ, а найденная последовательность из двух символов будет сохранена в переменной $1 за счет указания круглых скобок. Далее с помощью функции hex() мы преобразуем шестнадцатеричное число в десятичное, а затем получим символьное значение при помощи функции pack(). И, наконец, с помощью оператора s/// заменим все вхождения последовательностей %nn на соответствующие символы. Замена всех вхождений достигается за счет флага g, а флаг e указывает, что в параметре есть выражение языка Perl, которое необходимо предварительно вычислить.

3.3.2. Метод *POST*

Теперь рассмотрим технологию декодирования данных, переданных с помощью метода POST. Для этого изменим в коде нашей формы только значение параметра method с GET на POST (листинг 3.6).

Листинг 3.6. Обработка данных формы

```
<HTML>
<HEAD>
<TITLE>Обработка данных формы</TITLE>
</HEAD>
<BODY>
<B>Обработка данных формы</B>
<BR>
<FORM action="http://perlbook.ru/cgi-bin/form.pl" method="POST">
<INPUT type="text" name="txt1" size="20"><BR>
<INPUT type="text" name="txt2" size="20"><BR>
<INPUT type="submit" value="Передать">
</FORM>
</BODY>
</HTML>
```

Теперь напишем программу для декодирования данных этой формы. Причем в данном случае мы рассмотрим код, позволяющий декодировать данные, посланные как методом POST, так и методом GET. Откроем файл form.pl и заменим его содержимое на код из листинга 3.7.

Листинг 3.7. Декодирование данных формы

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use strict;
print "Content-type: text/html\n\n";

my @form_data;
our %MyForm;
my ($name, $value, $str);
if ($ENV{'REQUEST_METHOD'} eq 'GET') {
   # Данные переданы методом GET
   $str = $ENV{'QUERY_STRING'};
}
else {
   # Данные переданы методом POST
   read(STDIN, $str, $ENV{'CONTENT_LENGTH'});
}
@form_data = split(/&/, $str);
foreach (@form_data) {
   ($name, $value) = split(/=/, $_);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $MyForm{$name} = $value;
}
my ($key, $val);
while (($key, $val) = each(%MyForm)) {
   print "$key = $val<BR>";
}
```

В зависимости от метода передачи данных переменная окружения `$ENV{'REQUEST_METHOD'}` будет иметь значение `GET` или `POST`. Если данные переданы методом `GET`, то получим данные из переменной окружения `$ENV{'QUERY_STRING'}`:

```
   $str = $ENV{'QUERY_STRING'};
```

Если данные переданы методом POST, то читаем данные через стандартный ввод с помощью функции read() и сохраняем их в переменной $str. Количество переданных байт содержится в переменной окружения $ENV{'CONTENT_LENGTH'}:

```
read(STDIN, $str, $ENV{'CONTENT_LENGTH'});
```

Дальнейшее декодирование данных формы ничем не отличается от предыдущего примера.

3.3.3. Использование модуля *CGI*

При использовании модуля CGI.pm никакого декодирования данных формы производить не нужно. Достаточно указать имя поля в функции param(). Если данные были посланы, то функция вернет значение поля, а если нет, то функция вернет undef. Причем не важно, каким методом были посланы данные. Функция работает как с методом GET, так и с методом POST.

Модуль предоставляет два способа подключения. Первый способ заключается в импортировании стандартных функций с помощью тега :standard:

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);
print "Content-type: text/html\n\n";

print param("txt1") . "<BR>";
print param("txt2");
```

Второй способ использует объектно-ориентированный подход. В этом случае необходимо создать экземпляр класса с помощью метода new():

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI;
print "Content-type: text/html\n\n";
```

```
my $form = new CGI;
print $form->param("txt1") . "<BR>";
print $form->param("txt2");
```

Рассмотрим способы обработки данных каждого элемента формы по отдельности.

3.3.4. Текстовое поле, поле ввода пароля и скрытое поле

Отправим форму:

```
<INPUT type="text" name="txt">
<INPUT type="password" name="passw">
<INPUT type="hidden" name="hid" value="">
```

Мы можем получить значения полей следующими способами:

```
use CGI qw( :standard);
param("txt");
param("passw");
param("hid");
```

Или:

```
use CGI;
my $form = new CGI;
$form->param("txt");
$form->param("passw");
$form->param("hid");
```

Значение параметра функции `param()` совпадает со значением параметра `name` тега `<INPUT>`.

Предположим, что в эти поля необходимо ввести первоначальные значения в сценарии. В этом случае возможны следующие проблемы:

- Если в строке есть пробелы, то использование кавычек обязательно.

  Выведем так:

  ```
  #...
  my $str = "Привет всем";
  print "<INPUT type=\"text\" name=\"txt\" value=$str>";
  #...
  ```

  В результате поле `txt` будет содержать текст `"Привет"`, а не `"Привет всем"`. Правильно будет так:

```
#...
my $str = "Привет всем";
print "<INPUT type=\"text\" name=\"txt\" value=\"$str\">";
#...
```

- Если в строке есть кавычки, то их следует заменить на HTML-эквиваленты.

  Выведем так:

```
#...
my $str = "Группа \"Кино\"";
print "<INPUT type=\"text\" name=\"txt\" value=\"$str\">";
#...
```

  В результате поле `txt` будет содержать текст `"Группа "`, а не `"Группа "Кино""`. Правильно будет так:

```
#...
my $str = "Группа \"Кино\"";
$str =~ s/"/&quot;/g;
print "<INPUT type=\"text\" name=\"txt\" value=\"$str\">";
#...
```

3.3.5. Поле для ввода многострочного текста

Отправим форму:

```
<TEXTAREA name="txt">Текст</TEXTAREA>
```

Мы можем получить значение поля следующими способами:

```
use CGI qw( :standard);
param("txt");
```

Или:

```
use CGI;
my $form = new CGI;
$form->param("txt");
```

Значение параметра функции `param()` совпадает со значением параметра `name` тега `<TEXTAREA>`.

Предположим, что в это поле необходимо ввести первоначальные значения в сценарии:

```
#...
my $str = "Привет всем";
```

```
print "<TEXTAREA name=\"txt\">$str</TEXTAREA>";
#...
```

Если строка содержит теги, то их необходимо заменить на HTML-эквиваленты:

```
#...
my $str = "Привет </TEXTAREA>всем";
$str =~ s/&/&amp;/g;
$str =~ s/</&lt;/g;
$str =~ s/>/&gt;/g;
$str =~ s/"/&quot;/g;
print "<TEXTAREA name=\"txt\">$str</TEXTAREA>";
#...
```

3.3.6. Список с возможными значениями

Отправим форму:

```
<SELECT name="color">
<OPTION value="1">White
<OPTION>Red
</SELECT>
```

Мы можем получить значение элемента списка следующими способами:

```
use CGI qw( :standard);
my $color = param("color");
```

Или:

```
use CGI;
my $form = new CGI;
my $color = $form->param("color");
```

Значение параметра функции `param()` совпадает со значением параметра `name` тега `<SELECT>`.

Значение переменной будет присвоено в зависимости от выбранного значения в списке. Если выбран пункт **White**, то переменная `$color` будет иметь значение 1 (значение параметра `value`). Если выбран пункт **Red**, то переменная `$color` будет иметь значение `"Red"`, т. к. нет параметра `value`.

Если в списке можно выбрать сразу несколько значений, то все немного сложнее:

```
<SELECT name="day" size="7" MULTIPLE>
<OPTION value="1">Понедельник
```

```
<OPTION value="2">Вторник
<OPTION value="3">Среда
<OPTION value="4">Четверг
<OPTION value="5">Пятница
<OPTION value="6">Суббота
<OPTION value="7">Воскресенье
</SELECT>
```

Для получения всех выбранных значений необходимо на выходе функции `param()` указать массив:

```
use CGI qw( :standard);
my @day = param("day");
for(my $i=0; $i<@day; $i++) {
   print "$day[$i]<BR>";
}
```

Или:

```
use CGI;
my $form = new CGI;
my @day = $form->param("day");
for(my $i=0; $i<@day; $i++) {
   print "$day[$i]<BR>";
}
```

3.3.7. Флажок

Отправим форму:

```
<INPUT type="checkbox" name="check1" value="1"> Текст
<INPUT type="checkbox" name="check2"> Текст
```

Если флажки установлены, мы можем получить значения следующими способами:

```
use CGI qw( :standard);
my $check1 = param("check1");
my $check2 = param("check2");
```

Или:

```
use CGI;
my $form = new CGI;
my $check1 = $form->param("check1");
my $check2 = $form->param("check2");
```

Если флажки установлены, то переменные будут иметь следующие значения: переменная `$check1` — 1 (значение параметра `value`), а переменная `$check2` — `on` (нет параметра `value`).

> ***Внимание!***
> Если флажки не установлены, то значения не передаются!

По этой причине необходимо проверять существование значения поля:

```
use CGI qw( :standard);
if (defined(param("check1"))) {
   print "Флажок check1 установлен <BR>";
}
if (defined(param("check2"))) {
   print "Флажок check2 установлен";
}
```

Если флажки объединены в группу, то на выходе функции `param()` следует указать массив. Значение параметра `name` у всех флажков должно быть одинаковым, а значение параметра `value` — разным:

```
<INPUT type="checkbox" name="check" value="1"> Текст1
<INPUT type="checkbox" name="check" value="2"> Текст2
<INPUT type="checkbox" name="check" value="3"> Текст3

use CGI qw( :standard);
my @check;
if (defined(param("check"))) {
   @check = param("check");
   for(my $i=0; $i<@check; $i++) {
      print "$check[$i]<BR>";
   }
}
```

Или:

```
use CGI;
my $form = new CGI;
my @check;
if (defined($form->param("check"))) {
   @check = $form->param("check");
```

```
   for(my $i=0; $i<@check; $i++) {
      print "$check[$i]<BR>";
   }
}
```

3.3.8. Элемент-переключатель

Отправим форму:

```
<INPUT type="radio" name="pol" value="1" CHECKED> Мужской
<INPUT type="radio" name="pol" value="2"> Женский
```

Мы можем получить значение следующими способами:

```
use CGI qw( :standard);
my $pol = param("pol");
```

Или:

```
use CGI;
my $form = new CGI;
my $pol = $form->param("pol");
```

> ***Внимание!***
> Если ни один из переключателей не выбран, то значение не передается!

Переменная `$pol` будет иметь значение в зависимости от выбранного переключателя (1 или 2).

3.3.9. Кнопка *Submit*

Отправим форму:

```
<INPUT type="submit" name="go" value="Отправить">
```

Мы можем получить значение следующими способами:

```
use CGI qw( :standard);
if (defined(param("go"))) {
   print "Форма отправлена";
}
```

Или:

```
use CGI;
my $form = new CGI;
```

```
if (defined($form->param("go"))) {
   print "Форма отправлена";
}
```

Зачем для кнопки указывать параметр `name`? Все дело в том, что в одной форме может быть несколько кнопок **Submit**. Кроме того, если наш сценарий обрабатывает сразу несколько форм, то это позволит определить, какая форма отправлена.

Для этой цели может использоваться скрытое поле:

```
<FORM>
<INPUT type="radio" name="pol" value="1" CHECKED> Мужской
<INPUT type="radio" name="pol"> Женский
<INPUT type="hidden" name="go" value="send">
<INPUT type="submit" value="Отправить">
</FORM>
```

3.4. Работа с файлами и каталогами

Очень часто нужно сохранить какие-либо данные. Для этого существуют два способа:

- ❒ сохранение в файл;
- ❒ сохранение в базу данных.

Первый способ используется при сохранении информации небольшого объема. Если объем велик, то лучше (и удобнее) воспользоваться базой данных.

Файлы используются при создании гостевых книг, списков рассылки, ленты новостей, протоколирования различных ситуаций (например, ошибок) и во многих других случаях.

Для чтения или записи файла нужно выполнить следующие действия:

1. Открыть файл.
2. Блокировать файл.
3. Считать или записать данные.
4. Снять блокировку.
5. Закрыть файл.
6. Учитывая, что для ускорения работы производится буферизация данных, можно опустить явное снятие блокировки файла. Все дело в том, что информация из буфера записывается в файл только в момент закрытия фай-

ла. В период между снятием блокировки и закрытием файла другой процесс может успеть что-то записать в файл. При закрытии файла блокировка автоматически снимается.

3.4.1. Функции для работы с файлами

Рассмотрим основные функции для работы с файлами.

Функция `open(<Дескриптор>, '<Режим><Путь к файлу>')` открывает файл и возвращает дескриптор (идентификатор). При возникновении ошибки функция возвращает `false`, а описание ошибки сохраняется в специальной переменной `$!`. Параметр `<Режим>` может принимать следующие значения:

- `без указания режима` — только чтение. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция `open()` вернет `false`:

```
open(FILE, 'file1.txt') or die("Ошибка $!");
```

- `<` — только чтение. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция `open()` вернет `false`:

```
open(FILE, '<file1.txt') or die("Ошибка $!");
```

- `>` — запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла:

```
open(FILE, '>file1.txt') or die("Ошибка $!");
```

- `>>` — добавление. После открытия файла указатель устанавливается на конец файла. Если файл не существует, то он будет создан. Содержимое файла не удаляется:

```
open(FILE, '>>file1.txt') or die("Ошибка $!");
```

- `+<` — чтение и запись. Существующие данные не удаляются. Если в файле существуют данные, то запись в файл сразу после открытия будет сделана поверх существующих данных, т. к. после открытия файла указатель устанавливается на начало файла. По этой причине не забудьте изменить положение указателя при записи:

```
open(FILE, '+<file1.txt') or die("Ошибка $!");
```

- `+>` — чтение и запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла:

```
open(FILE, '+>file1.txt') or die("Ошибка $!");
```

- `+>>` — чтение и запись. Запись выполняется в конец файла. Если файл не существует, то он будет создан. Существующие данные не удаляются:

```
open(FILE, '+>>file1.txt') or die("Ошибка $!");
```

Функция `close(<Дескриптор>)` закрывает файл:

```
close(FILE);
```

Функция `flock(<Дескриптор>, <Режим>)` позволяет блокировать файл. Параметр `<Режим>` может принимать следующие значения:

- `LOCK_EX` (или `2`) — монопольный режим. Файл не доступен для совместного использования;
- `LOCK_UN` (или `8`) — снимает блокировку:

```
open(FILE, '>>file1.txt') or die("Ошибка $!");
flock(FILE, 2) or die("Ошибка $!");
print FILE "строка";
flock(FILE, 8) or die("Ошибка $!");
close(FILE);
```

Для использования символьных значений необходимо подключить модуль `Fcntl`:

```
use Fcntl qw( :flock );
open(FILE, '>>file1.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
print FILE "строка";
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Оператор `print <Дескриптор> "<Что пишем>"` записывает данные в файл:

```
print FILE "строка";
```

Оператор `<>` ("ромб") позволяет считывать данные из файла до тех пор, пока не будет найден символ, который совпадает со значением переменной `$/`. По умолчанию значение переменной `$/` равно `\n`. Иными словами, оператор `<>` считывает данные из файла построчно. Для примера сохраним несколько строк в файл, а затем выведем их на экран (листинг 3.8).

Листинг 3.8. Работа с файлами

```
use Fcntl qw( :flock );

open(FILE1, '>>file1.txt') or die("Ошибка $!");
flock(FILE1, LOCK_EX) or die("Ошибка $!");
```

```
print FILE1 "строка1\n";
print FILE1 "строка2\n";
flock(FILE1, LOCK_UN) or die("Ошибка $!");
close(FILE1);

open(FILE2, 'file1.txt') or die("Ошибка $!");
flock(FILE2, LOCK_EX) or die("Ошибка $!");
while (<FILE2>) {
   print $_, "<BR>";
}
flock(FILE2, LOCK_UN) or die("Ошибка $!");
close(FILE2);
```

Если необходимо считать весь файл в строку, то специальной переменной `$/` необходимо присвоить пустую строку (листинг 3.9).

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Следует помнить, что изменение значения специальной переменной может привести к ошибкам в дальнейшем. Чтобы избежать этого, необходимо внутри блока объявлять динамическую переменную с помощью ключевого слова `local`. В этом случае после выхода из блока значение специальной переменной останется прежним.

Листинг 3.9. Считывание всего файла в строку

```
use Fcntl qw( :flock );
my $text;

open(FILE, 'file1.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
{
   local $/ = "";
   $text = <FILE>;
}
print "<PRE>", $text, "</PRE>";
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Если на выходе оператора <> указать массив, то весь файл будет построчно считан в массив (листинг 3.10).

Листинг 3.10. Считывание всего файла в массив

```
use Fcntl qw( :flock );
my @text;

open(FILE, 'file1.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
@text = <FILE>;
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
for(my $i=0; $i<@text; $i++) {
   print "$text[$i]<BR>";
}
```

При считывании файла в массив следует помнить, что каждый элемент массива содержит символ переноса строки. Чтобы от него избавиться, необходимо воспользоваться функцией `chomp()`:

```
chomp(@text);
```

Функция `read(<Дескриптор>, <Переменная>, <Длина в байтах>)` позволяет прочитать из файла строку указанной длины и сохранить ее в переменной (листинг 3.11).

Листинг 3.11. Функция `read()`

```
use Fcntl qw( :flock );
my $text;

open(FILE, 'file1.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
read(FILE, $text, 6);
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
print "$text<BR>";
# Выведет строка
```

Функция `getc(<Дескриптор>)` позволяет считывать из файла по одному символу за раз (листинг 3.12).

Листинг 3.12. Функция `getc()`

```
use Fcntl qw( :flock );
my $text;
```

```
open(FILE, 'file1.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
while ($text = getc(FILE)) {
   print "$text";
}
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Функция `sysopen(<Дескриптор>, <Путь к файлу>, <Режим>, [<Права доступа>])` открывает файл и возвращает дескриптор. В отличие от функции `open()` она позволяет задать права доступа (в виде числа) создаваемому файлу, а также задать отдельные компоненты режима работы с файлом. Параметр `<Режим>` может принимать комбинацию следующих значений:

- `O_RDONLY` — только для чтения;
- `O_WRONLY` — только для записи;
- `O_RDWR` — для чтения и записи;
- `O_CREAT` — создать файл, если он не существует;
- `O_EXCL` — не открывать файл, если он существует;
- `O_APPEND` — добавить в конец файла;
- `O_TRUNC` — очистить содержимое файла;
- `O_NONBLOCK` — неограниченный доступ.

Указанные значения импортируются из модуля `Fcntl`:

```
use Fcntl;
```

В качестве примера приведем соответствие режимов функций `open()` и `sysopen()`:

- `<` — `O_RDONLY`:

```
sysopen(FILE, 'file1.txt', O_RDONLY) or die("Ошибка $!");
```

- `>` — `O_WRONLY | O_CREAT | O_TRUNC`:

```
sysopen(FILE, 'file1.txt', O_WRONLY | O_CREAT | O_TRUNC)
 or die("Ошибка $!");
```

- `>>` — `O_WRONLY | O_CREAT | O_APPEND`:

```
sysopen(FILE, 'file1.txt', O_WRONLY | O_CREAT | O_APPEND)
 or die("Ошибка $!");
```

- `+<` — `O_RDWR`:

```
sysopen(FILE, 'file1.txt', O_RDWR) or die("Ошибка $!");
```

- **+>** — O_RDWR | O_CREAT | O_TRUNC:

```
sysopen(FILE, 'file1.txt', O_RDWR | O_CREAT | O_TRUNC)
or die("Ошибка $!");
```

- **+>>** — O_RDWR | O_CREAT | O_APPEND:

```
sysopen(FILE, 'file1.txt', O_RDWR | O_CREAT | O_APPEND)
or die("Ошибка $!");
```

Для примера сохраним несколько строк в файл, а затем выведем их на экран (листинг 3.13).

Листинг 3.13. Функция sysopen()

```
use Fcntl qw( :DEFAULT :flock );

sysopen(FILE1, 'file2.txt', O_RDWR | O_CREAT | O_APPEND, 0644)
or die("Ошибка $!");
flock(FILE1, LOCK_EX) or die("Ошибка $!");
print FILE1 "строка1\n";
print FILE1 "строка2\n";
flock(FILE1, LOCK_UN) or die("Ошибка $!");
close(FILE1);

sysopen(FILE2, 'file2.txt', O_RDONLY) or die("Ошибка $!");
flock(FILE2, LOCK_EX) or die("Ошибка $!");
while (<FILE2>) {
   print $_, "<BR>";
}
flock(FILE2, LOCK_UN) or die("Ошибка $!");
close(FILE2);
```

Рассмотрим еще один пример. Создадим файл file.txt и запишем в него пять E-mail-адресов по одному в строке (листинг 3.14).

Листинг 3.14. Создание файла и запись в него

```
use Fcntl qw( :flock );

my $mail = "mail1\@site.ru\nmail2\@site.ru\nmail3\@site.ru\n";
$mail .= "mail4\@site.ru\nmail5\@site.ru";
```

```
open(FILE, ">file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
print FILE $mail;
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
print "Файл создан";
```

Теперь добавим новую запись в конец файла (листинг 3.15).

Листинг 3.15. Добавление новой записи в конец файла

```
use Fcntl qw( :flock );

my $mail = "\nmail6\@site.ru";
open(FILE, ">>file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
print FILE $mail;
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
print "Операция произведена";
```

А теперь выведем содержимое файла в список (листинг 3.16).

Листинг 3.16. Вывод содержимого файла в список

```
use Fcntl qw( :flock );

my @text;
open(FILE, "file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
@text = <FILE>;
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
# Выводим содержимое файла в список
chomp(@text);
print "<SELECT>\n";
for(my $i=0; $i<@text; $i++) {
   print "<OPTION>", $text[$i], "\n";
}
print "</SELECT>\n";
```

3.4.2. Перемещение внутри файла

Рассмотрим основные функции для перемещения и манипулирования позицией указателя внутри файла.

Функция `tell(<Дескриптор>)` возвращает позицию указателя относительно начала файла (листинг 3.17).

Листинг 3.17. Функция `tell()`

```
use Fcntl qw( :flock );

open(FILE, ">>file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
print tell(FILE);
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Функция `seek(<Дескриптор>, <Смещение>, <Позиция>)` устанавливает указатель в позицию, имеющую смещение `<Смещение>` относительно позиции `<Позиция>`. Параметр `<Позиция>` может принимать следующие значения:

- `SEEK_SET` (или `0`) — начало файла;
- `SEEK_CUR` (или `1`) — текущая позиция указателя;
- `SEEK_END` (или `2`) — конец файла.

Для установки указателя на конец файла можно воспользоваться кодом из листинга 3.18.

Листинг 3.18. Установка указателя на конец файла

```
use Fcntl qw( :flock );

open(FILE, "+<file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
seek(FILE, 0, 2);
print tell(FILE);
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Для использования символьных названий необходимо подключить модуль `IO::Seekable`. В качестве примера откроем файл для добавления и переместим указатель на начало файла (листинг 3.19).

Листинг 3.19. Перемещение указателя на начало файла

```
use Fcntl qw( :flock );
use IO::Seekable;

open(FILE, ">>file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
seek(FILE, 0, SEEK_SET);
print tell(FILE);
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
```

Функция `truncate(<Дескриптор>, <Позиция>)` удаляет все данные от позиции `<Позиция>` до конца файла (листинг 3.20).

Листинг 3.20. Функция `truncate()`

```
use Fcntl qw( :flock );
my $text;

open(FILE, "+>file.txt") or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
print FILE "строка1\n";
print FILE "строка2\n";
truncate(FILE, 7);
seek(FILE, 0, 0);
{
local $/ = "";
$text = <FILE>;
}
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);
print "<PRE>", $text, "</PRE>";
# Выведет: строка1
```

3.4.3. Права доступа в операционной системе UNIX

Большинство хостинговых площадок используют операционную систему семейства UNIX. В этой ОС для каждого объекта (файла или каталога) назна-

чаются права доступа для каждой разновидности пользователей — владельца, группы и прочих.

Для файла или каталога могут быть назначены следующие права доступа:

- чтение;
- запись;
- выполнение.

Права доступа обозначаются буквами:

- `r` — файл можно читать, а содержимое каталога можно просматривать;
- `w` — файл можно модифицировать, удалять и переименовывать, а в каталоге можно создавать или удалять файлы. Каталог можно переименовать или удалить;
- `x` — файл можно выполнять, а в каталоге можно выполнять операции над файлами, в том числе производить поиск файлов в нем.

Права доступа к файлу определяются записью типа:

```
-rw-r--r--
```

Первый символ (`-`) означает, что это файл и не задает никаких прав доступа. Далее три символа (`rw-`) задают права доступа для владельца (чтение и запись). Символ `-` означает, что нет права доступа на выполнение. Следующие три символа задают права доступа для группы (`r--`) — только чтение. Ну и последние три символа (`r--`) задают права для всех остальных пользователей (только чтение).

Права доступа к каталогу определяются такой строкой:

```
drwxr-xr-x
```

Первая буква (`d`) означает, что это каталог. Владелец может выполнять в каталоге любые действия (`rwx`), а группа и все остальные пользователи — только читать и выполнять поиск (`r-x`). Для того чтобы каталог можно было просматривать, должны быть установлены права на выполнение (`x`).

Кроме того, права доступа обозначаются числом. Такие числа называются *маской прав доступа*. Число состоит из трех цифр от 0 до 7. Первая цифра задает права для владельца, вторая — для группы, а третья — для всех остальных пользователей. Например, права доступа `-rw-r--r--` соответствуют числу 644.

Переведем числа в двоичный формат. Рассмотрим, например, права доступа:

```
rw-r--r--
```

Можно записать так:

```
110 100 100
```

Это соответствует числу:

```
6 4 4
```

Таким образом, право установлено — это 1, а если нет — это 0.

Для файлов, не являющихся CGI-программами, таких как html, shtml или php — права доступа могут быть установлены в 644 (`-rw-r--r--`) (запись-чтение для владельца и только чтение для всех остальных).

Для файлов, являющихся CGI-программами (Perl-скрипты, скомпилированные C-программы и прочие), права доступа должны быть установлены в 755 (`rwxr-xr-x`) (исполнение-запись-чтение для владельца и чтение-исполнение для всех остальных).

Права доступа на каталоги рекомендуется устанавливать в 755 (`rwxr-xr-x`).

Чтобы изменить права доступа из скрипта, необходимо воспользоваться функцией `chmod()`. Функция имеет следующий формат:

```
chmod(<Права доступа>, <Путь к файлу>);
```

Права доступа задаются в виде числа, перед которым следует указать 0:

```
chmod(0644, $path);
```

С помощью функции `chown()` можно изменить права собственности файла. Функция имеет следующий формат:

```
chown(<ID пользователя>, <ID группы>, <Путь к файлу>);
```

В операционной системе UNIX каждый пользователь и группа имеют уникальные идентификационные номера. В функции `chown()` ID пользователя и ID группы указываются именно в виде числа.

3.4.4. Файловые проверки

Для проверки существования файла используются следующие операторы:

- `-e <Путь к файлу>` — возвращает `true`, если файл существует:

```
my $file = "file.txt";
if (-e $file) {
   print "Файл существует";
}
else {
   print "Файл НЕ существует";
}
```

- `-z <Путь к файлу>` — возвращает `true`, если файл существует и имеет нулевую длину:

```
my $file = "file.txt";
if (-z $file) {
   print "Файл существует и имеет нулевую длину";
}
```

- `-s <Путь к файлу>` — если файл существует и имеет ненулевую длину, то оператор возвращает размер файла в байтах:

```
my $file = "file1.txt";
if (-s $file) {
   print "Размер файла ", -s $file, " байт";
}
```

Для определения прав доступа можно использовать следующие операторы:

- `-r <Путь к файлу>` — возвращает `true`, если файл доступен для чтения:

```
my $file = "file.txt";
if (-r $file) {
   print "Файл доступен для чтения";
}
else {
   print "Файл НЕ доступен для чтения";
}
```

- `-w <Путь к файлу>` — возвращает `true`, если файл доступен для записи:

```
my $file = "file.txt";
if (-w $file) {
   print "Файл доступен для записи";
}
else {
   print "Файл НЕ доступен для записи";
}
```

- `-x <Путь к файлу>` — возвращает `true`, если файл является исполняемым:

```
my $file = "file.txt";
if (-x $file) {
   print "Файл является исполняемым";
}
else {
   print "Файл НЕ является исполняемым";
}
```

Для определения типа файла предназначены операторы:

- `-f <Путь к файлу>` — возвращает `true`, если файл является простым (не каталог и не символическая ссылка):

```
my $file = "file1.txt";
if (-f $file) {
   print "Файл является простым";
}
```

- `-d <Путь к файлу>` — возвращает `true`, если это каталог:

```
my $file = "folder";
if (-d $file) {
   print "Это каталог";
}
```

- `-l <Путь к файлу>` — возвращает `true`, если это символическая ссылка;
- `-b <Путь к файлу>` — возвращает `true`, если файл является специальным блочным файлом;
- `-c <Путь к файлу>` — возвращает `true`, если это специальный символический файл;
- `-T <Путь к файлу>` — возвращает `true`, если файл является текстовым;
- `-B <Путь к файлу>` — возвращает `true`, если файл является двоичным.

Кроме того, существуют операторы для определения возраста файла, времени последнего доступа и модификации файла:

- `-C <Путь к файлу>` — возвращает количество дней с момента создания файла:

```
my $file = "file.txt";
print -C $file;
```

- `-A <Путь к файлу>` — возвращает количество дней с момента последнего доступа к файлу:

```
my $file = "file.txt";
print -A $file;
```

- `-M <Путь к файлу>` — возвращает количество дней с момента последней модификации файла:

```
my $file = "file.txt";
print -M $file;
```

Для получения информации о файле можно воспользоваться функцией `stat()`, которая возвращает массив значений, соответствующих структуре

индексного дескриптора в файловой системе UNIX. Функция возвращает массив следующих значений:

- 0 — номер устройства;
- 1 — номер индексного дескриптора;
- 2 — режим файла;
- 3 — количество жестких ссылок на файл;
- 4 — ID владельца файла;
- 5 — ID группы владельца файла;
- 6 — идентификатор устройства;
- 7 — размер файла в байтах;
- 8 — время последнего доступа к файлу (количество секунд, прошедших с 1 января 1970 г.);
- 9 — время последнего изменения файла (количество секунд, прошедших с 1 января 1970 г.);
- 10 — время с момента создания файла (количество секунд, прошедших с 1 января 1970 г.);
- 11 — размер блока для операций ввода и вывода;
- 12 — количество блоков для размещения файла.

Пример функции:

```
my @file_stat = stat("file.txt");
for(my $i=0; $i<@file_stat; $i++) {
   print "$i – $file_stat[$i]<BR>";
}
```

3.4.5. Функции для манипулирования файлами

Функция `rename(<Старое имя>, <Новое имя>)` переименовывает файл. Если новое имя файла уже существует, то функция перезапишет файл с новым именем и содержимое файла будет потеряно:

```
rename('file.txt', 'file1.txt') or die("Ошибка $!");
```

Функция `unlink(<Путь к файлу>)` позволяет удалить файл или несколько файлов. Функция возвращает количество успешно удаленных файлов:

```
my $count = unlink('file1.txt') or die("Ошибка $!");
print $count;
```

Функция `utime(<Время последнего доступа к файлу>, <Время модификации файла>, <Название файла>)` устанавливает для файла (или файлов) время последнего доступа и время последнего изменения файла. Временные параметры задаются в виде количества секунд, прошедших с 1 января 1970 г. Функция возвращает количество успешно модифицированных файлов:

```
my $ctime = time() — 86400;
my $count = utime($ctime, $ctime, 'file.txt') or die("Ошибка $!");
print $count;
```

Функция `glob(<Строка с шаблоном>)` возвращает файлы, имена которых соответствуют заданному шаблону. В шаблоне можно использовать символы `*` и `?`. Для примера создадим в каталоге со скриптом три текстовых файла — file1.txt, file2.txt и file3.txt. Теперь произведем поиск всех файлов с расширением TXT:

```
my @files = glob("*.txt");
for(my $i=0; $i<@files; $i++) {
   print "$files[$i]<BR>";
}
```

Вывод:

```
file1.txt
file2.txt
file3.txt
```

Вместо использования функции `glob()` можно указать шаблон в угловых скобках. В итоге получим такой же результат:

```
my @files = <*.txt>;
for(my $i=0; $i<@files; $i++) {
   print "$files[$i]<BR>";
}
```

Функция `binmode(<Дескриптор>)` подготавливает файл для записи или чтения в двоичном формате:

```
binmode(FILE);
```

3.4.6. Загрузка файлов на сервер

Загрузка файлов на сервер осуществляется с помощью multipart-формы. Создадим файл file_load.html в каталоге C:\WebServers\home\perlbook.ru\www с кодом из листинга 3.21.

Листинг 3.21. Содержимое файла file_load.html

```
<HTML>
<HEAD>
<TITLE>Загрузка файлов</TITLE>
</HEAD>
<BODY>
<B>Загрузка файлов</B>
<BR>
<FORM action="http://perlbook.ru/cgi-bin/file.pl" method="POST"
enctype="multipart/form-data">
<INPUT type="file" name="file_name" size="20">
<INPUT type="submit" value="Загрузить">
</FORM>
</BODY>
</HTML>
```

Далее создаем файл file.pl в папке C:\WebServers\home\perlbook.ru\cgi-bin и добавляем в него код из листинга 3.22.

Листинг 3.22. Содержимое файла file.pl

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
# подключаем модуль для обработки данных формы
use CGI qw( :standard);

my $path = "C:\\WebServers\\home\\perlbook.ru\\www\\";
my $file = param('file_name');
# Выделяем имя файла
$file =~ m/([^\/\\:]+)$/;
my $name_file = $1;
my $mypath = $path . $name_file;
# Пишем в файл
open(FILE, ">$mypath") or die("Ошибка $!");
binmode(FILE);
```

```
print FILE <$file>;
close(FILE);

print "Content-type: text/html\n\n";
if (-e $mypath) {
   print "Файл успешно загружен";
}
else {
   print "Ошибка при загрузке файла";
}
```

При выборе файла с помощью кнопки **Обзор...** и нажатии кнопки **Загрузить** файл будет отправлен серверу. Отправка будет выполнена только в том случае, если параметр `enctype` тега `<FORM>` имеет значение `"multipart/form-data"`:

```
enctype="multipart/form-data"
```

Содержимое отправленного файла доступно через автоматически создаваемый дескриптор, название которого совпадает с именем файла.

Рассмотрим программу из файла file.pl подробно. В первой строке как обычно указывается путь к интерпретатору Perl:

```
#!/usr/bin/perl -w
```

Далее подключаем модуль `CGI::Carp`, который позволяет увидеть все ошибки в окне Web-браузера:

```
use CGI::Carp qw(fatalsToBrowser);
```

Для обработки данных формы подключаем модуль `CGI`. При помощи `qw( :standard)` мы можем обратиться к значению элемента формы, просто указав его имя в функции `param()`:

```
use CGI qw( :standard);
```

Затем в переменной `$path` сохраняем путь к папке, в которую будем загружать файл:

```
my $path = "C:\\WebServers\\home\\perlbook.ru\\www\\";
```

Следует обратить внимание на символ, разделяющий путь. В операционных системах Windows и UNIX этот символ будет разным. В ОС Windows используется символ \, а в ОС UNIX — символ /.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Символ \ является специальным. По этой причине мы его экранируем.

В следующей строке мы получаем значение поля `file_name` с помощью функции `param()` и сохраняем его в переменной `$file`:

```
my $file = param('file_name');
```

Далее с помощью регулярного выражения выделяем имя файла и сохраняем его в переменной `$name_file`:

```
$file =~ m/([^\/\\:]+)$/;
my $name_file = $1;
```

Для удобства в переменной `$mypath` сохраняем полный путь к файлу:

```
my $mypath = $path . $name_file;
```

Затем открываем файл на запись. Если файл не существует, то он будет автоматически создан. Если файл уже существует, то содержимое файла будет очищено:

```
open(FILE, ">$mypath") or die("Ошибка $!");
```

В следующей строке с помощью функции `binmode()` подготавливаем файл для записи в двоичном формате:

```
binmode(FILE);
```

Далее пишем в файл:

```
print FILE <$file>;
```

Для записи двоичного файла можно использовать функцию `read()`. В этом случае предыдущее выражение следует заменить на следующий код:

```
while (read($file, $buffer, 1024)) {
   print FILE $buffer;
}
```

После записи файла закрываем его:

```
close(FILE);
```

С помощью строки:

```
print "Content-type: text/html\n\n";
```

устанавливаем заголовки ответа сервера, а затем выводим результат загрузки файла. Если файл существует, то оператор `-e` возвращает значение `true`:

```
if (-e $mypath) {
   print "Файл успешно загружен";
}
else {
   print "Ошибка при загрузке файла";
}
```

3.4.7. Функции для работы с каталогами

Для работы с каталогами используются следующие функции:

- `mkdir(<Имя каталога>, <Права доступа>)` — создает новый каталог с правами доступа, указанными во втором параметре. Права доступа указываются в виде трехзначного числа, перед которым указывается 0. Например, 0755;
- `rmdir(<Имя каталога>)` — удаляет пустой каталог. Если в каталоге есть файлы, то каталог удален не будет;
- `chdir(<Имя каталога>)` — делает указанный каталог текущим;
- `opendir(<Дескриптор>, <Имя каталога>)` — открывает каталог для чтения. Функция возвращает дескриптор, который указывается в других функциях;
- `readdir(<Дескриптор>)` — считывает следующее имя объекта (файла или подкаталога);
- `closedir(<Дескриптор>)` — закрывает каталог;
- `rewinddir(<Дескриптор>)` — перемещает указатель в начало каталога.

Прочитаем содержимое каталога C:\WebServers\home\perlbook.ru\www и выведем содержимое каталога в окно Web-браузера (листинг 3.23), причем каталоги и файлы выведем отдельно, а для файлов укажем дату создания, дату последнего доступа и дату последней модификации файла.

Листинг 3.23. Чтение каталога

```
#!/usr/bin/perl -w
# Выводим все сообщения об ошибках
# в окно Web-браузера
use CGI::Carp qw(fatalsToBrowser);
use strict;
use POSIX;
# Настройка локали
use locale;
setlocale(LC_ALL, 'ru_RU.CP1251');
print "Content-type: text/html\n\n";

my (@file, @dir, $file_name);
my $path = "C:\\WebServers\\home\\perlbook.ru\\www\\";
chdir($path); # Делам каталог текущим
```

```
opendir(DIR, $path) or die("Ошибка $!"); # Открываем каталог
foreach(readdir(DIR)) { # Читаем каталог
   $file_name = $path . $_;
   if (-d $file_name) {
      push(@dir, $_);
   }
   if (-f $file_name) {
      push(@file, $_);
   }
}
closedir(DIR); # Закрываем каталог

# Выводим каталоги
print "<B>Каталоги</B><BR>\n";
print "<UL>\n";
for(my $i=0; $i<@dir; $i++) {
   print "<LI>$dir[$i]\n";
}
print "</UL>\n";

# Выводим файлы
print "<B>Файлы</B><BR>\n";
print "<UL>\n";
for(my $i=0; $i<@file; $i++) {
   print "<LI>$file[$i]\n";
   my @file_stat = stat($path . $file[$i]);
   print "<UL>\n";
   print "<LI>Создан — " . f_time_file($file_stat[10]) . "\n";
   print "<LI>Последний доступ — " . f_time_file($file_stat[8]) . "\n";
   print "<LI>Последняя модификация — ";
   print f_time_file($file_stat[9]) . "\n";
   print "</UL>\n";
}
print "</UL>\n";

# Функция для форматирования даты
sub f_time_file {
```

```
    my $s = shift();
    return strftime("%d.%m.%Y %H:%M:%S", localtime($s));
}
```

Сохраняем файл под именем dir.pl в каталоге C:\WebServers\home \perlbook.ru\cgi-bin, а затем открываем Web-браузер и в адресной строке набираем:

```
http://perlbook.ru/cgi-bin/dir.pl
```

В результате отобразится содержимое каталога C:\WebServers\home \perlbook.ru\www.

3.5. Получение информации из Интернета

В предыдущем разделе мы рассмотрели методику работы с локально сохраненными файлами. Теперь изучим возможности Perl для работы со Всемирной паутиной. В Perl есть множество модулей, объединенных в единую библиотеку `LWP` (`libwww-perl`). Эта библиотека позволяет автоматизировать многие задачи, связанные с работой в Интернете.

3.5.1. Модуль *LWP::Simple*

Для начала рассмотрим самый простой способ получения документа из Интернета. Модуль `LWP::Simple` предоставляет процедурный интерфейс и позволяет получить документ методом `GET`, а также посмотреть заголовки с помощью метода `HEAD`.

Модуль содержит следующие функции:

- `get(<URL-адрес>)` — возвращает документ по указанному URL-адресу. В случае неудачи функция возвращает значение `undef`:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::Simple;
my $doc = get("http://wwwadmin.ru/");
if (defined($doc)) {
   print $doc;
}
```

```
else {
   print "Не удалось получить документ";
}
```

- `head(<URL-адрес>)` — возвращает заголовки документа в виде списка. В скалярном контексте при удачном запросе функция возвращает `true`. Элементы списка содержат следующие значения:
  - `0` — тип документа (заголовок `Content-Type`);
  - `1` — размер документа (заголовок `Content-Length`);
  - `2` — дата последнего изменения файла (заголовок `Last-Modified`);
  - `3` — заголовок `Expires`;
  - `4` — описание сервера (заголовок `Server`):

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::Simple;
my @doc = head("http://wwwadmin.ru/");
if (@doc) {
   for(my $i=0; $i<@doc; $i++) {
      print "$i — $doc[$i]<BR>";
   }
}
else {
   print "Не удалось получить документ";
}
```

- `getprint(<URL-адрес>)` — получает документ и печатает его на `STDOUT`. Возвращает код возврата HTTP:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::Simple;
my $status = getprint("http://wwwadmin.ru/");
print "<BR><BR>Код возврата ", $status;
```

- `getstore(<URL-адрес>, <Имя файла>)` — получает документ и сохраняет его в файл. Возвращает код возврата HTTP:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::Simple;
my $status = getstore("http://wwwadmin.ru/", "file.txt");
print "Код возврата ", $status;
```

Все указанные функции в качестве агента пользователя возвращают строку:

```
LWP::Simple/<Номер версии>
```

Например:

```
LWP::Simple/5.805
```

Если необходимо изменить эту информацию, то следует импортировать экземпляр класса `LWP::UserAgent` из модуля, указав его в списке импорта:

```
use LWP::Simple qw/ :DEFAULT $ua/;
```

Теперь с помощью метода `agent()` класса `LWP::UserAgent` меняем информацию об агенте пользователя:

```
$ua->agent("MySpider/1.0");
```

Для примера получим документ, предварительно указав свое название агента пользователя (листинг 3.24).

Листинг 3.24. Получение документа из Интернета

```
use LWP::Simple qw/ :DEFAULT $ua/;
$ua->agent("MySpider/1.0");
$doc = get("http://wwwadmin.ru/");
if (defined($doc)) {
   print $doc;
}
else {
   print "Не удалось получить документ";
}
```

3.5.2. Класс *HTTP::Request*

Класс `HTTP::Request` описывает запрос клиента к серверу. Класс имеет следующие методы:

- `new(<Метод>, <URL-адрес>)` — создает новый экземпляр класса `HTTP::Request`. В параметре `<Метод>` могут быть указаны значения `GET`, `POST`, `HEAD` или `PUT`. Метод может принимать ссылку на экземпляр класса `HTTP::Headers` в качестве третьего необязательного параметра:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->agent("MySpider/1.0");
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
```

  Параметры `<Метод>` и `<URL-адрес>` можно передать с помощью методов `method(<Метод>)` и `uri(<URL-адрес>)` соответственно:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->agent("MySpider/1.0");
my $request = HTTP::Request->new();
$request->method(GET);
$request->uri('http://wwwadmin.ru/');
```

- `as_string()` — возвращает заголовки сформированного запроса в виде строки. Используется для отладки:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->agent("MySpider/1.0");
```

```
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
print $request->as_string();
```

Вывод:

```
GET http://wwwadmin.ru/
```

В данном примере мы не получили заголовок:

```
User-Agent: MySpider/1.0
```

Это произошло потому, что мы не передавали этот заголовок экземпляру класса `HTTP::Request`:

- `header(<Заголовок> => <Значение>,..., <Заголовок> => <Значение>)` — позволяет добавить заголовок или изменить значение существующего. Методу можно передать сразу несколько пар `<Заголовок> => <Значение>` за один раз. Метод `header()` наследуется от класса `HTTP::Headers`. В предыдущих примерах мы устанавливали название нашего робота с помощью метода `agent()` класса `LWP::UserAgent`. Теперь рассмотрим добавление заголовка идентифицирующего нашего робота с помощью метода `header()`:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
print $request->as_string();
```

Вывод:

```
GET http://wwwadmin.ru/
Accept: text/html, text/plain
User-Agent: MySpider/1.0
```

В этом примере мы добавили два заголовка — `Accept` (предпочитаемые MIME-типы) и `User-Agent` (название нашего робота).

Если в качестве параметра передать методу `header()` только название заголовка, то метод возвратит текущее значение этого заголовка:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
print $request->header('Accept');
```

Вывод:

```
text/html, text/plain
```

- `push_header(<Заголовок> => <Значение>)` — позволяет добавить новое значение к указанному заголовку. Метод `push_header()` наследуется от класса `HTTP::Headers`:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
$request->push_header('Accept' => ", image/*");
print $request->header('Accept');
```

Вывод:

```
text/html, text/plain, image/*
```

- `remove_header(<Заголовок>)` — позволяет удалить указанный заголовок. Метод `remove_header()` наследуется от класса `HTTP::Headers`:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
$request->remove_header('Accept');
print $request->as_string();
```

Вывод:

```
GET http://wwwadmin.ru/
User-Agent: MySpider/1.0
```

3.5.3. Класс *HTTP::Response*

Класс `HTTP::Response` описывает ответ сервера на запрос. Класс имеет следующие методы:

- `is_success()` — возвращает `true`, если запрос был успешным;
- `status_line()` — возвращает код ответа и его описание:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
```

```
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print "Запрос выполнен успешно. ", $result->status_line();
}
else {
   print "Неудачный запрос. ", $result->status_line();
}
```

Вывод:

```
Запрос выполнен успешно. 200 OK
```

Или при отсутствии документа:

```
Неудачный запрос. 404 Not Found
```

- `code()` — возвращает код ответа в виде числа;
- `message()` — возвращает описание кода ответа:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print "Запрос выполнен успешно. ";
   print $result->code(), " — ", $result->message();
}
else {
   print "Неудачный запрос. ";
   print $result->code(), " — ", $result->message();
}
```

Вывод:

```
Запрос выполнен успешно. 200 — OK
```

Или при отсутствии документа:

```
Неудачный запрос. 404 — Not Found
```

А при отсутствии сайта по указанному адресу:

```
Неудачный запрос. 500 — Can't connect to wwwadmin.ru:80 (Bad hostname
'wwwadmin.ru')
```

- `protocol()` — возвращает название протокола:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print $result->protocol();
}
```

Вывод:

```
HTTP/1.1
```

- `is_error()` — возвращает `true` в случае ошибки;
- `error_as_HTML()` — возвращает HTML-документ, описывающий сообщение об ошибке:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin2.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_error()) {
   print $result->error_as_HTML();
}
```

Вывод:

```
<HTML>
<HEAD><TITLE>An Error Occurred</TITLE></HEAD>
<BODY>
<H1>An Error Occurred</H1>
500 Can't connect to wwwadmin2.ru:80 (Bad hostname 'wwwadmin2.ru')
</BODY>
</HTML>
```

- `base()` — возвращает базовый URL-адрес для данного документа. Базовый URL-адрес получается на основе анализа тега `<BASE>`, заголовков `Content-Base` и `Content-Location`, а также обработки перенаправлений:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://kino-x.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print  $result->base();
}
```

Вывод:

```
http://kino-x.ru/kino-x/index.asp
```

- `as_string()` — возвращает текстовое представление результата запроса (все заголовки ответа сервера плюс сам документ):

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print $result->as_string();
}
```

- `content()` — возвращает содержимое ответа сервера без заголовков:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print  $result->content();
}
```

- `headers_as_string()` — возвращает заголовки ответа сервера, а также информацию из тегов `<TITLE>` и `<META>`:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print  $result->headers_as_string();
}
```

Вывод:

```
Connection: close
Date: Tue, 01 Jul 2008 15:53:26 GMT
Server: nginx/0.5.35
Content-Type: text/html; charset=windows-1251
Content-Type: text/html; charset=windows-1251
Client-Date: Tue, 01 Jul 2008 15:53:27 GMT
Client-Peer: 89.111.176.111:80
Client-Response-Num: 1
Client-Transfer-Encoding: chunked
Title: Учебники и самоучители по HTML, CSS, JavaScript, Apache, PHP,
MySQL
X-Meta-Description: В книге рассматриваются вопросы создания
Web-сайтов с помощью HTML, JavaScript, PHP и MySQL без использования
специализированных редакторов...
X-Meta-Keywords: Учебники и самоучители по HTML, CSS, JavaScript,
Apache, PHP, MySQL
X-Meta-Robots: index, follow
X-Powered-By: PHP/4.3.9
```

- `header(<Заголовок>)` — возвращает значение указанного заголовка ответа сервера. В качестве примера выведем название документа и его описание:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
```

```
my $result = $robot->request($request);
if ($result->is_success()) {
   print  $result->header('Title'), "<BR>";
   print  $result->header('X-Meta-Description');
}
```

- `content_type()` — возвращает значение заголовка `Content-Type` в нижнем регистре. В скалярном контексте позволяет проверить MIME-тип документа следующим образом:

```
   if ($result->content_type eq 'text/html') {
      print "Это HTML-документ";
   }
```

Пример:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   my @contentType = $result->content_type();
   print  $contentType[0], "<BR>";
   print  $contentType[1], "<BR>";
   if ($result->content_type eq 'text/html') {
      print "Это HTML-документ";
   }
}
```

Вывод:

```
text/html
charset=windows-1251
Это HTML-документ
```

3.5.4. Класс *LWP::UserAgent*

Класс `LWP::UserAgent` эмулирует агента пользователя и объединяет сразу несколько классов: `HTTP::Request`, `HTTP::Response` и `LWP::Protocol`. Класс имеет следующие методы:

- `new()` — создает новый экземпляр класса `LWP::UserAgent`:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
```

- `agent()` — позволяет задать название робота:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->agent("MySpider/1.0");
```

- `request(<Экземпляр класса HTTP::Request>)` — обрабатывает запрос, учитывая перенаправления:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://kino-x.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
print $result->status_line();
```

Вывод:

```
200 OK
```

- `simple_request(<Экземпляр класса HTTP::Request>)` — обрабатывает одиночный запрос. Перенаправления автоматически не обрабатываются:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://kino-x.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->simple_request($request);
print $result->status_line();
print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
```

Вывод:

```
302 Object moved
Cache-Control: private
```

```
Date: Tue, 01 Jul 2008 19:37:58 GMT
Location: kino-x/index.asp
Server: Microsoft-IIS/6.0
Content-Length: 137
Content-Type: text/html
Client-Date: Tue, 01 Jul 2008 19:40:06 GMT
Client-Peer: 88.212.201.100:80
Client-Response-Num: 1
Title: Object moved
X-Powered-By: ASP.NET
```

- `max_size(<Размер>)` — позволяет установить предельный размер содержимого отклика. По умолчанию предельный размер отсутствует. Если предел превышен, то в отклик добавляется заголовок:

```
Client-Aborted: max_size
```

Возвращается только указанное количество байтов:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->max_size(10);
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print $result->status_line();
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
}
```

Вывод:

```
200 OK
Connection: close
Date: Thu, 03 Jul 2008 15:51:48 GMT
Server: nginx/0.5.35
Content-Type: text/html; charset=windows-1251
Client-Aborted: max_size
Client-Date: Thu, 03 Jul 2008 15:51:47 GMT
Client-Peer: 89.111.176.111:80
Client-Response-Num: 1
Client-Transfer-Encoding: chunked
```

```
Title: Учебники и самоучители по HTML, CSS, JavaScript, Apache, PHP,
MySQL
X-Meta-Description: В книге рассматриваются вопросы создания
Web-сайтов с помощью HTML, JavaScript, PHP и MySQL без использования
специализированных редакторов...
X-Powered-By: PHP/4.3.9
```

- `timeout(<Количество секунд>)` — позволяет установить максимальное время обработки запроса. По умолчанию установлено значение 180 секунд (3 минуты);
- `parse_head(<1 | 0>)` — определяет, нужно ли производить автоматически разбор раздела HEAD HTML-документа. По умолчанию разбор производится:

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
$robot->parse_head(0);
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print $result->status_line();
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
}
```

Вывод:

```
Connection: close
Date: Thu, 03 Jul 2008 16:05:52 GMT
Server: nginx/0.5.35
Content-Type: text/html; charset=windows-1251
Client-Date: Thu, 03 Jul 2008 16:05:52 GMT
Client-Peer: 89.111.176.111:80
Client-Response-Num: 1
Client-Transfer-Encoding: chunked
X-Powered-By: PHP/4.3.9
```

Как видно из примера, мы не получили заголовки `Title`, `X-Meta-Description`, `X-Meta-Keywords` и `X-Meta-Robots`;

- `max_redirect(<Число>)` — позволяет задать максимальное количество обрабатываемых перенаправлений при использовании метода `request()`. По умолчанию установлено значение 7.

3.5.5. Класс *LWP::RobotUA*

Класс `LWP::RobotUA` позволяет роботу автоматически учитывать требования файла robots.txt и ограничивает слишком частые запросы к одному серверу. Помните: роботы должны с уважением относиться к серверам, которые они посещают.

Класс `LWP::RobotUA` наследует все методы класса `LWP::UserAgent` и предоставляет следующие дополнительные методы:

- `new(<Название робота>, <E-mail владельца робота>)` — создает новый экземпляр класса `LWP::RobotUA`:

```
use LWP::RobotUA;
my $email = "unicross\@mail.ru";
my $robot = LWP::RobotUA->new("MySpider/1.0", $email);
```

- `delay(<Количество минут>)` — устанавливает минимальную задержку между запросами к одному и тому же серверу. Значение по умолчанию — 1 минута. Если необходимо задать значение в секундах, то указывается следующее значение:

```
$robot->delay(10/60);
```

  В этом примере мы установили задержку на 10 секунд:

```
use LWP::RobotUA;
my $email = "unicross\@mail.ru";
my $robot = LWP::RobotUA->new("MySpider/1.0", $email);
$robot->delay(10/60);
```

- `use_sleep(<1 | 0>)` — задает поведение робота при ограничении запроса методом `delay()`. Если установлено значение 1 (по умолчанию), то робот будет ожидать окончания установленного периода, прежде чем отправить запрос. Если установлено значение 0, то возвращается заголовок `Retry-After`, значение которого показывает, когда будет разрешено отправить еще один запрос данному серверу:

```
use LWP::RobotUA;
my $email = "unicross\@mail.ru";
my $robot = LWP::RobotUA->new("MySpider/1.0", $email);
$robot->delay(2);
$robot->use_sleep(0);
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('Accept' => "text/html, text/plain");
my $result = $robot->request($request);
```

```
print $result->status_line();
print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
```

В данном примере мы отказываемся от ожидания и при простой перезагрузке страницы получим следующий ответ:

```
503 Please, slow down
Retry-After: Thu, 03 Jul 2008 17:03:38 GMT
```

Если запрашиваемый ресурс запрещен для индексации в файле robots.txt, то получим ответ:

```
403 Forbidden by robots.txt
```

Теперь рассмотрим получение документов методами `HEAD`, `GET` и `POST`.

3.5.6. Метод *HEAD*

Метод `HEAD` позволяет получить только заголовки ответа сервера (листинг 3.25). Применяется он для определения работоспособности ресурса. При выполнении `HEAD`-запросов используется меньше ресурсов и времени.

Листинг 3.25. Метод `HEAD`

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(HEAD => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
my $result = $robot->request($request);
if ($result->is_success()) {
   print $result->status_line();
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

Вывод:

```
200 OK
Connection: close
Date: Thu, 03 Jul 2008 17:21:39 GMT
Server: nginx/0.5.35
```

```
Content-Type: text/html; charset=windows-1251
Client-Date: Thu, 03 Jul 2008 17:21:38 GMT
Client-Peer: 89.111.176.111:80
Client-Response-Num: 1
Client-Transfer-Encoding: chunked
X-Powered-By: PHP/4.3.9
```

Кроме того, можно воспользоваться методом `head()` класса `LWP::UserAgent`.

```
head(<URL-адрес>, <Заголовки>)
```

Рассмотрим метод (листинг 3.26).

Листинг 3.26. Метод `head()`

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $result = $robot->head('http://wwwadmin.ru/',
 'User-Agent' => "MySpider/1.0",
 'Accept' => "text/html, text/plain");
if ($result->is_success()) {
   print $result->status_line();
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

3.5.7. Метод *GET*

Метод `GET` является самым распространенным методом получения документов из Интернета. В листинге 3.27 приведен пример получения документа методом `GET`.

Листинг 3.27. Метод `GET`

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(GET => 'http://wwwadmin.ru/');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
```

```
my $result = $robot->request($request);
print $result->status_line();
if ($result->is_success()) {
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
   print "<BR><PRE>", $result->content(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

Кроме того, можно воспользоваться методом `get()` класса `LWP::UserAgent`.

```
get(<URL-адрес>, <Заголовки>)
```

Рассмотрим метод (листинг 3.28).

Листинг 3.28. Метод `get()`

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my @MyHeaders = (
 'User-Agent' => "MySpider/1.0",
 'Accept' => "text/html, text/plain"
);
my $result = $robot->get('http://wwwadmin.ru/', @MyHeaders);
print $result->status_line();
if ($result->is_success()) {
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
   print "<BR><PRE>", $result->content(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

3.5.8. Метод *POST*

Метод `POST` используется для передачи данных большого размера, а также конфиденциальных данных. Для примера инсценируем передачу данных формы для публикации сообщений в гостевой книге (листинг 3.29).

Листинг 3.29. Метод POST

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my $request = HTTP::Request->new(POST =>
'http://wwwadmin.ru/catalog/gbook.php');
$request->header('User-Agent' => "MySpider/1.0",
'Accept' => "text/html, text/plain");
$request->content_type('application/x-www-form-urlencoded');
$request->content('i_author=Nik&i_msg=Hello&i_go=go');
my $result = $robot->request($request);
print $result->status_line();
if ($result->is_success()) {
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
   print "<BR><PRE>", $result->content(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

С помощью метода `content_type()` мы указываем тип содержимого `application/x-www-form-urlencoded`, которое задает передачу данных формы. Далее с помощью метода `content()` мы передаем сами данные.

Кроме того, можно воспользоваться методом `post()` класса `LWP::UserAgent`:

```
post(<URL-адрес>, [<Данные формы>], <Заголовки>)
```

Рассмотрим метод (листинг 3.30).

Листинг 3.30. Метод post()

```
use LWP::UserAgent;
my $robot = LWP::UserAgent->new();
my @MyHeaders = (
 'User-Agent' => "MySpider/1.0",
 'Accept' => "text/html, text/plain"
);
my $result = $robot->post('http://wwwadmin.ru/catalog/gbook.php',
['i_author'=>'Nik',
'i_msg'=>'Hello',
'i_go'=>'go'],
```

```
 'User-Agent' => "MySpider/1.0",
 'Accept' => "text/html, text/plain"
);
print $result->status_line();
if ($result->is_success()) {
   print "<BR><PRE>", $result->headers_as_string(), "</PRE>";
   print "<BR><PRE>", $result->content(), "</PRE>";
}
else {
   print "Ресурс недоступен";
}
```

Итак, мы научились получать документ из Интернета. Теперь рассмотрим синтаксический разбор HTML-документа, а также его составных частей.

3.5.9. Разбор URL-адреса

С помощью модуля `URI` мы можем манипулировать URL-адресом. Например, можно разобрать его на составляющие или получить абсолютный URL-адрес, указав базовый адрес и относительный. Модуль `URI` предоставляет следующие методы:

- `new(<URL-адрес>)` — позволяет создать экземпляр класса:

```
use URI;
my $path = "http://www.wwwadmin.ru:80/test.php?var=5#metka";
my $url = URI->new($path);
```

- `as_string()` — возвращает текстовое представление объекта. Добавим к предыдущему коду строку:

```
print $url->as_string();
```

  Вывод:

```
http://www.wwwadmin.ru:80/test.php?var=5#metka
```

- `scheme()` — возвращает название протокола:

```
print $url->scheme();
```

  Вывод:

```
http
```

- `opaque()` — возвращает текстовое представление объекта без названия протокола и без "якоря":

```
print $url->opaque();
```

Примечание

"Якорем" называют метку внутри HTML-документа. Эта метка позволяет быстро перемещаться внутри большого документа от одного раздела к другому с помощью гиперссылок, в которых название метки указывается после символа #.

Вывод:

```
//www.wwwadmin.ru:80/test.php?var=5
```

- `authority()` — возвращает название хоста вместе с номером порта:

```
print $url->authority();
```

Вывод:

```
www.wwwadmin.ru:80
```

- `host()` — возвращает название хоста:

```
print $url->host();
```

Вывод:

```
www.wwwadmin.ru
```

- `port()` — возвращает номер порта:

```
print $url->port();
```

Вывод:

```
80
```

- `path()` — возвращает путь:

```
print $url->path();
```

Вывод:

```
/test.php
```

- `query()` — возвращает строку запроса:

```
print $url->query();
```

Вывод:

```
var=5
```

- `path_query()` — возвращает путь и строку запроса:

```
print $url->path_query();
```

Вывод:

```
/test.php?var=5
```

- `fragment()` — возвращает "якорь":

```
print $url->fragment();
```

Вывод:

```
metka
```

3.5.10. Преобразование относительной ссылки в абсолютную

Очень часто в HTML-документах указываются не абсолютные ссылки, а относительные. При относительном URL-адресе путь определяется с учетом местоположения Web-страницы, на которой находится ссылка. Модуль `URI` позволяет преобразовать относительную ссылку в абсолютный URL-адрес:

```
use URI;
local $URI::ABS_REMOTE_LEADING_DOTS = 1;
my $url = URI->new(<Относительный адрес>)->abs(<Абсолютный адрес>);
my $url_abs = $url->as_string();
```

Кроме того, можно воспользоваться методом `new_abs()`. Метод имеет следующий формат:

```
use URI;
my $url = URI->new_abs(<Относительный адрес>, <Абсолютный адрес>);
my $url_abs = $url->as_string();
```

Например:

```
use URI;
my $abs_url = "http://wwwadmin.ru/f1/f2/test.html";
my $url = URI->new_abs("file.html", $abs_url);
print $url->as_string();
```

Вывод:

```
http://wwwadmin.ru/f1/f2/file.html
```

Рассмотрим различные относительные ссылки более подробно:

```
use URI;
local $URI::ABS_REMOTE_LEADING_DOTS = 1;
my $abs_url = "http://wwwadmin.ru/f1/f2/test.html";
print "Абсолютный URL — " . $abs_url . "<BR>";
my $url = URI->new("file.html")->abs($abs_url);
print "'file.html' -> " . $url->as_string() . "<BR>";
```

```
$url = URI->new("/file.html")->abs($abs_url);
print "'/file.html' -> " . $url->as_string() . "<BR>";
$url = URI->new("./file.html")->abs($abs_url);
print "'./file.html' -> " . $url->as_string() . "<BR>";
$url = URI->new("../file.html")->abs($abs_url);
print "'../file.html' -> " . $url->as_string() . "<BR>";
$url = URI->new("../../file.html")->abs($abs_url);
print "'../../file.html' -> " . $url->as_string() . "<BR>";
$url = URI->new("../../../file.html")->abs($abs_url);
print "'../../../file.html' -> " . $url->as_string() . "<BR>";
```

Вывод:

```
Абсолютный URL — http://wwwadmin.ru/f1/f2/test.html
'file.html' -> http://wwwadmin.ru/f1/f2/file.html
'/file.html' -> http://wwwadmin.ru/file.html
'./file.html' -> http://wwwadmin.ru/f1/f2/file.html
'../file.html' -> http://wwwadmin.ru/f1/file.html
'../../file.html' -> http://wwwadmin.ru/file.html
'../../../file.html' -> http://wwwadmin.ru/file.html
```

В последнем случае мы специально указали уровень относительности больше, чем нужно. Никакой ошибки в данном случае не возникает.

3.5.11. Кодирование и декодирование URL-адреса

Модуль `URI::Escape` позволяет преобразовать специальные символы, входящие в состав URL-адреса, в последовательность `%nn`. И, наоборот, позволяет преобразовать последовательности `%nn` в символы.

Функция `uri_escape()` преобразует специальные символы в последовательность `%nn`:

```
use URI::Escape;
print uri_escape("Строка параметров");
```

Вывод:

```
%D1%F2%F0%EE%EA%E0%20%EF%E0%F0%E0%EC%E5%F2%F0%EE%E2
```

Функция `uri_unescape()` преобразует последовательности `%nn` в символы:

```
use URI::Escape;
my $text = "%D1%F2%F0%EE%EA%E0%20%EF%E0%F0%E0%EC%E5%F2%F0%EE%E2";
print uri_unescape($text);
```

Вывод:

```
Строка параметров
```

3.5.12. Разбор HTML-эквивалентов

В HTML-документе ряд символов являются специальными. Например, знак "меньше" (<) и знак "больше" (>), символы кавычек и амперсанда. Для отображения специальных символов используются так называемые HTML-эквиваленты. Например, знак "меньше" заменяется на последовательность `<`. Модуль `HTML::Entities` позволяет манипулировать HTML-эквивалентами:

- `encode_entities(<Строка>, [<Список символов>])` — заменяет специальные символы на HTML-эквиваленты. По умолчанию заменяются все символы, в том числе и символы русского алфавита. Параметр `<Список символов>` позволяет указать конкретные символы для замены;
- `decode_entities(<Строка>)` — заменяет HTML-эквиваленты на обычные символы.

Для примера заменим специальные символы на HTML-эквиваленты (листинг 3.31).

Листинг 3.31. Разбор HTML-эквивалентов

```
use HTML::Entities;
my $str = '<A href="file.html">Ссылка</A>';
my $str1 = encode_entities($str, '<>"&');
print $str1, "<BR>\n";
my $str2 = decode_entities($str);
print $str2;
```

Вывод:

```
&lt;A href=&quot;file.html&quot;&gt;Ссылка&lt;/A&gt;<BR>
<A href="file.html">Ссылка</A>
```

3.5.13. Преобразование кодировок

Документы в Интернете могут быть в различных кодировках. Модуль `Text::Iconv` позволяет преобразовать текст из одной кодировки в другую. Модуль не входит в состав ActivePerl, и нет возможности установить его с сайта **http://www.activestate.com/**. Для установки модуля `Text::Iconv`

запускаем Денвер. Переходим на диск Z и запускаем файл Z:\usr \local\perl\bin\ppm-shell.bat. В командной строке должно быть приглашение:

```
ppm>
```

Набираем команду:

```
install http://theoryx5.uwinnipeg.ca/ppms/Text-Iconv.ppd
```

и нажимаем клавишу <Enter>.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Файл ppm-shell.bat необходимо обязательно запустить с диска Z. В противном случае модуль будет установлен в каталог C:\usr, а не в C:\WebServers\usr.

Преобразование кодировок выполняется следующим образом:

```
use Text::Iconv;
my $iconv = Text::Iconv->new(<Исходная кодировка>, <Нужная кодировка>);
print $iconv->convert(<Строка>);
```

Пример преобразования кодировок приведен в листинге 3.32.

Листинг 3.32. Преобразование кодировок

```
use Text::Iconv;
my  $iconv1 = Text::Iconv->new("windows-1251", "koi8-r");
print $iconv1->convert("Строка"); # Выведет уФТПЛБ
print "<BR>";
my  $iconv2 = Text::Iconv->new("windows-1251", "UTF-8");
print $iconv2->convert("Строка");
print "<BR>";
my  $iconv3 = Text::Iconv->new("koi8-r", "windows-1251");
print $iconv3->convert("уФТПЛБ"); # Выведет Строка
```

3.5.14. Модуль *HTML::LinkExtor*

Модуль `HTML::LinkExtor` позволяет извлечь из HTML-документа все ссылки. Возвращает ссылки не только из тега `<A>`, но и из тегов `<IMG>`, `<FRAME>` и `<AREA>`.

Метод `new(<Ссылка на подпрограмму>, [<Базовый URL-адрес>])` позволяет создать экземпляр класса. Если указан `<Базовый URL-адрес>`, то все относительные ссылки будут преобразованы в абсолютные:

```
use HTML::LinkExtor;
my $base_url = "http://wwwadmin.ru/folder1/index.html";
```

```
my $parser = HTML::LinkExtor->new(\&f_parser, $base_url);
sub f_parser {
   my($tag, %links) = @_;
   # Подпрограмма обработки
}
```

Метод `parse(<HTML-текст>)` запускает обработку HTML-документа, переданного в качестве параметра. Рассмотрим получение всех ссылок из HTML-документа. Выведем отдельно ссылки по тегам и отделим ссылки на E-mail-адрес от обычных ссылок (листинг 3.33).

Листинг 3.33. Получение всех ссылок из HTML-документа

```
use HTML::LinkExtor;

my $HTML_text = <<HTMLCode;
<HTML><HEAD>
<TITLE>Заголовок страницы</TITLE></HEAD>
<BODY>
<A href="../file1.html" alt="Описание">Ссылка</A><BR>
<!--
<A href="file2.html">Ссылка</A><BR>
-->
<A href="mailto:mail\@mysite.ru">Текст</A>
<!-- Изображения -->
<IMG SRC="img.gif" width="8"><BR>
<!-- Фреймы -->
<FRAMESET rows="100, *">
<FRAME src="doc1.html">
</FRAMESET>
<!-- Карты-изображения -->
<MAP name="karta">
<AREA share="rect" coords="0,0,120,120" href="chapter1.html"
target="chapter" alt="Глава 1">
<AREA share="rect" coords="0,120,240,240" href="chapter2.html"
target="chapter" alt="Глава 2">
<AREA shape="default" nohref>
</MAP>
</BODY></HTML>
HTMLCode
```

```
my $base_url = "http://wwwadmin.ru/folder1/index.html";
our (@link, @mailto, @img, @frame, @map);
my $parser = HTML::LinkExtor->new(\&f_parser, $base_url);
$parser->parse($HTML_text);

if (defined(@link)) {
   print "<LI><B>Ссылки:</B><BR>";
   for(my $i=0; $i<@link; $i++) {
      print "$link[$i]<BR>";
   }
}
if (defined(@mailto)) {
   print "<LI><B>Ссылки на E-mail:</B><BR>";
   for(my $i=0; $i<@mailto; $i++) {
      print "$mailto[$i]<BR>";
   }
}
if (defined(@img)) {
   print "<LI><B>Изображения:</B><BR>";
   for(my $i=0; $i<@img; $i++) {
      print "$img[$i]<BR>";
   }
}
if (defined(@frame)) {
   print "<LI><B>Фреймы:</B><BR>";
   for(my $i=0; $i<@frame; $i++) {
      print "$frame[$i]<BR>";
   }
}
if (defined(@map)) {
   print "<LI><B>Карты-изображения:</B><BR>";
   for(my $i=0; $i<@map; $i++) {
      print "$map[$i]<BR>";
   }
}

sub f_parser {
   my($tag, %links) = @_;
```

```
   if ($tag eq 'a') {
      if ($links{'href'} =~ /^mailto:/i) {
         push(@mailto, $links{'href'}) ;
      }
      else {
         push(@link, $links{'href'}) ;
      }
   }
   elsif ($tag eq 'img') {
      push(@img, $links{'src'}) ;
   }
   elsif ($tag eq 'frame') {
      push(@frame, $links{'src'}) ;
   }
   elsif ($tag eq 'area') {
      push(@map, $links{'href'}) ;
   }
}
```

Вывод:

```
Ссылки:
http://wwwadmin.ru/file1.html
Ссылки на E-mail:
mailto:mail@mysite.ru
Изображения:
http://wwwadmin.ru/folder1/img.gif
Фреймы:
http://wwwadmin.ru/folder1/doc1.html
Карты-изображения:
http://wwwadmin.ru/folder1/chapter1.html
http://wwwadmin.ru/folder1/chapter2.html
```

Как видно из примера, все относительные ссылки были преобразованы в абсолютные. Обратите внимание: если ссылка содержится внутри символов HTML-комментария, то мы не получим ее значения.

Метод `parse_file(<Имя файла>)` запускает обработку HTML-документа, содержащегося в указанном файле. Для примера создадим два файла. Файл HTMLtext.txt (листинг 3.34) будет содержать обрабатываемый HTML-текст, а второй файл будет содержать программу обработки (листинг 3.35).

Листинг 3.34. Содержимое файла HTMLtext.txt

```
<HTML><HEAD>
<TITLE>Заголовок страницы</TITLE></HEAD>
<BODY>
<A href="../file1.html" alt="Описание">Ссылка</A><BR>
<!--
<A href="file2.html">Ссылка</A><BR>
-->
<A href="mailto:mail@mysite.ru">Текст</A>
<!-- Изображения -->
<IMG SRC="img.gif" width="8"><BR>
<!-- Фреймы -->
<FRAMESET rows="100, *">
<FRAME src="doc1.html">
</FRAMESET>
<!-- Карты-изображения -->
<MAP name="karta">
<AREA share="rect" coords="0,0,120,120" href="chapter1.html"
target="chapter" alt="Глава 1">
<AREA share="rect" coords="0,120,240,240" href="chapter2.html"
target="chapter" alt="Глава 2">
<AREA shape="default" nohref>
</MAP>
</BODY></HTML>
```

Листинг 3.35. Текст программы обработки

```
use HTML::LinkExtor;

my $base_url = "http://wwwadmin.ru/folder1/index.html";
our (@link, @mailto, @img, @frame, @map);
my $parser = HTML::LinkExtor->new(\&f_parser, $base_url);
$parser->parse_file("HTMLtext.txt");

sub f_parser {
   my($tag, %links) = @_;
     print "$tag => @{[%links]}<BR>\n";
}
```

Вывод:

```
a => href http://wwwadmin.ru/file1.html
a => href mailto:mail@mysite.ru
img => src http://wwwadmin.ru/folder1/img.gif
frame => src http://wwwadmin.ru/folder1/doc1.html
area => href http://wwwadmin.ru/folder1/chapter1.html
area => href http://wwwadmin.ru/folder1/chapter2.html
```

Метод `links()` возвращает все ссылки из HTML-документа (листинг 3.36).

Листинг 3.36. Метод `links()`

```
use HTML::LinkExtor;

my $HTML_text = <<HTMLCode;
<HTML><HEAD>
<TITLE>Заголовок страницы</TITLE></HEAD>
<BODY>
<A href="../file1.html" alt="Описание">Ссылка</A><BR>
<!--
<A href="file2.html">Ссылка</A><BR>
-->
<A href="mailto:mail\@mysite.ru">Текст</A>
<!-- Изображения -->
<IMG SRC="img.gif" width="8"><BR>
<!-- Фреймы -->
<FRAMESET rows="100, *">
<FRAME src="doc1.html">
</FRAMESET>
<!-- Карты-изображения -->
<MAP name="karta">
<AREA share="rect" coords="0,0,120,120" href="chapter1.html"
target="chapter" alt="Глава 1">
<AREA share="rect" coords="0,120,240,240" href="chapter2.html"
target="chapter" alt="Глава 2">
<AREA shape="default" nohref>
</MAP>
</BODY></HTML>
HTMLCode
```

```
my $base_url = "http://wwwadmin.ru/folder1/index.html";
my $parser = HTML::LinkExtor->new(undef, $base_url);
$parser->parse($HTML_text);
my @all_links = $parser->links();
foreach my $item (@all_links) {
   print $item->[0], " => ", $item->[1], " => ", $item->[2], "<BR>";
}
```

Вывод:

```
a => href => http://wwwadmin.ru/file1.html
a => href => mailto:mail@mysite.ru
img => src => http://wwwadmin.ru/folder1/img.gif
frame => src => http://wwwadmin.ru/folder1/doc1.html
area => href => http://wwwadmin.ru/folder1/chapter1.html
area => href => http://wwwadmin.ru/folder1/chapter2.html
```

3.5.15. Модуль *HTML::TokeParser*

Модуль `HTML::TokeParser` позволяет произвести полный синтаксический разбор HTML-документа.

Метод `new(<Ссылка на HTML-код или Имя файла>)` позволяет создать экземпляр класса:

```
use HTML::TokeParser;
my $parser = HTML::TokeParser->new(\$HTML_text);
my $parser2 = HTML::TokeParser->new("file.txt");
```

Метод `get_token()` возвращает из HTML-документа очередной маркер. В зависимости от типа маркера метод возвращает следующие значения:

- `S` — открывающий тег. Значения массива:
  - `0` — тип маркера (`S`);
  - `1` — имя тега;
  - `2` — ассоциативный массив параметров тега;
  - `3` — список названий параметров тега;
  - `4` — исходный текст тега;
- `E` — закрывающий тег. Значения массива:
  - `0` — тип маркера (`E`);
  - `1` — имя тега;
  - `2` — исходный текст тега;

- `T` — текст. Значения массива:
  - `0` — тип маркера (`T`);
  - `1` — исходный текст;
- `C` — комментарий. Значения массива:
  - `0` — тип маркера (`C`);
  - `1` — исходный текст комментария.

В качестве примера произведем синтаксический разбор HTML-документа (листинг 3.37).

Листинг 3.37. Синтаксический разбор HTML-документа

```
use HTML::TokeParser;
use HTML::Entities;

my $HTML_text = <<HTMLCode;
<HTML><HEAD>
<TITLE>Заголовок страницы</TITLE></HEAD>
<BODY>
<!-- Заголовок -->
<H1>Заголовок первого уровня</H1>
<!-- Ссылка -->
<A href="../file1.html" alt="Описание">Ссылка</A><BR>
<!--
Комментарий
-->
<!-- Изображения -->
<IMG SRC="img.gif" width="468"><BR>
</BODY></HTML>
HTMLCode

print "----------Начальный тег---------------------------------<BR>\n";
my $parser = HTML::TokeParser->new(\$HTML_text);
while (my $item = $parser->get_token()) {
   if ($item->[0] eq 'S') {
      print "<B>", $item->[1], "</B><BR>";
      print "  ", encode_entities($item->[4], '<>"&');
```

```
      print "<BR>\n";
      my @attr = @{$item->[3]};
      for (my $i=0; $i<@attr; $i++) {
         print "  ", $attr[$i], " => ", $item->[2]{$attr[$i]};
         print "<BR>\n";
      }
   }
}
print "---------Конечный тег----------------------------------<BR>\n";
my $parser2 = HTML::TokeParser->new(\$HTML_text);
while (my $item = $parser2->get_token()) {
   if ($item->[0] eq 'E') {
      print "<B>", $item->[1], "</B><BR>";
      print "  ", encode_entities($item->[2], '<>"&');
      print "<BR>\n";
   }
}
print "----------Текст-----------------------------------------<BR>\n";
my $parser3 = HTML::TokeParser->new(\$HTML_text);
while (my $item = $parser3->get_token()) {
   if ($item->[0] eq 'T') {
      if ($item->[1] ne '' && $item->[1] ne "\n") {
         print $item->[1], "<BR>\n";
      }
   }
}
print "----------Комментарий-----------------------------------<BR>\n";
my $parser4 = HTML::TokeParser->new(\$HTML_text);
while (my $item = $parser4->get_token()) {
   if ($item->[0] eq 'C') {
         print encode_entities($item->[1], '<>"&'), "<BR>\n";
   }
}
```

Вывод:

```
----------Начальный тег---------------------------------
html
  <HTML>
```

```
head
  <HEAD>
title
  <TITLE>
body
  <BODY>
h1
  <H1>
a
  <A href="../file1.html" alt="Описание">
  href => ../file1.html
  alt => Описание
br
  <BR>
img
  <IMG SRC="img.gif" width="468">
  src => img.gif
  width => 468
br
  <BR>
---------Конечный тег----------------------------------
title
  </TITLE>
head
  </HEAD>
h1
  </H1>
a
  </A>
body
  </BODY>
html
  </HTML>
----------Текст-----------------------------------------
Заголовок страницы
Заголовок первого уровня
Ссылка
```

```
----------Комментарий----------------------------------
<!-- Заголовок -->
<!-- Ссылка -->
<!-- Комментарий -->
<!-- Изображения -->
```

Метод `get_tag(<Список тегов>)` возвращает маркеры, соответствующие указанным тегам. Возвращает массив следующих значений для начального тега:

- `0` — имя тега;
- `1` — ассоциативный массив параметров тега;
- `2` — список названий параметров тега;
- `3` — исходный текст тега.

Для получения текста между начальным и конечным тегами можно воспользоваться методами `get_text(<Конечный тег>)` и `get_trimmed_text(<Конечный тег>)`. Отличие между двумя методами заключается в том, что метод `get_trimmed_text()` удаляет начальные и конечные пробельные символы, а пробельные символы в середине строки заменяет на пробел. Например, при разборе ссылки:

```
<A href="file.html">
Ссылка Строка1
Ссылка Строка2
</A>
```

Метод `get_text()` вернет значение:

```
Ссылка Строка1
Ссылка Строка2
```

А метод `get_trimmed_text()` вернет:

```
Ссылка Строка1 Ссылка Строка2
```

Для примера разберем теги `<TITLE>`, `<A>` и `<IMG>` (листинг 3.38).

Листинг 3.38. Разбор HTML-документа

```
use HTML::TokeParser;
use HTML::Entities;

my $HTML_text = <<HTMLCode;
<HTML><HEAD>
<TITLE>Заголовок страницы</TITLE></HEAD>
```

```
<BODY>
<A href="../file1.html" alt="Описание">Текст ссылки</A><BR>
<IMG SRC="img.gif" width="468"><BR>
</BODY></HTML>
HTMLCode

my $parser = HTML::TokeParser->new(\$HTML_text);
my @tags = ("a", "img", "title");
while (my $item = $parser->get_tag(@tags)) {
   if ($item->[0] eq 'title') {
      print "title<BR>  ";
      print $parser->get_trimmed_text("/title");
   }
   if ($item->[0] eq 'a') {
      print "\n<BR>a<BR>";
      print "  ", encode_entities($item->[3], '<>"&');
      print "<BR>\n";
      my @attr = @{$item->[2]};
      for (my $i=0; $i<@attr; $i++) {
         print "  ", $attr[$i], " => ", $item->[1]{$attr[$i]};
         print "<BR>\n";
      }
      print "  ", $parser->get_trimmed_text("/a");
   }
   if ($item->[0] eq 'img') {
      print "\n<BR>img<BR>";
      print "  ", encode_entities($item->[3], '<>"&');
      print "<BR>\n";
      my @attr = @{$item->[2]};
      for (my $i=0; $i<@attr; $i++) {
         print "  ", $attr[$i], " => ", $item->[1]{$attr[$i]};
         print "<BR>\n";
      }
   }
}
```

Вывод:

```
title
  Заголовок страницы
```

```
a
  <A href="../file1.html" alt="Описание">
  href => ../file1.html
  alt => Описание
  Текст ссылки
img
  <IMG SRC="img.gif" width="468">
  src => img.gif
  width => 468
```

3.6. Отправка писем с сайта

Отправить письма с сайта позволяет программа `Sendmail`. Технология отправки писем выглядит следующим образом:

```
open(<Дескриптор>, "| <Путь к программе> <Флаг>") or die("Ошибка $!");
print <Дескриптор> "<Заголовки>\n\n";
print <Дескриптор> "<Текст письма>";
close(<Дескриптор>);
```

С помощью функции `open()` мы создаем дескриптор файла. Символ | указывает, что созданный дескриптор связан с конвейером. Далее указывается путь к программе `Sendmail`. Обычно такой:

```
/usr/sbin/sendmail
```

В параметре `<Флаг>` можно указать ключ. Обычно указывается ключ `-t`, который означает, что заголовки передаются в тексте сообщения. От текста сообщения заголовки отделяются с помощью двух символов переноса строки (`\n\n`).

В параметре `<Заголовки>` обычно указываются следующие заголовки:

- `From` — имя и обратный адрес отправителя:

  ```
  From: Nik <unicross@mail.ru>
  ```

- `To` — имя и адрес получателя:

  ```
  To: Ivan <ivan@mail.ru>
  ```

- `Subject` — тема сообщения:

  ```
  Subject: Test mail
  ```

- `Content-Type` — MIME-тип и кодовая таблица:

  ```
  Content-Type: text/html; charset=windows-1251
  ```

- `Content-Transfer-Encoding` — количество битов, которое используется для передачи символа. Для русского языка следует указать значение `8bit`:

  ```
  Content-Transfer-Encoding: 8bit
  ```

Любое письмо может быть отправлено в виде обычного текста, а также в формате HTML. В первом случае указывается MIME-тип `text/plain`, а во втором — `text/html`.

В качестве примера отправим письмо с подтверждением регистрации в виде обычного текста (листинг 3.39), а также в формате HTML (листинг 3.40).

Листинг 3.39. Пример письма в виде обычного текста

```
my $header = "From: support <support\@site.ru>\n";
$header .= "To: Ivan <ivan\@mail.ru>\n";
$header .= "Subject: Test mail\n";
$header .= "Content-Type: text/plain; charset=windows-1251\n";
$header .= "Content-Transfer-Encoding: 8bit\n\n";

my $msg = "Добрый день!\n\n";
$msg .= "Вы успешно зарегистрированы.\n\n";
$msg .= "http://www.site.ru/\n";
$msg .= "support\@site.ru";

open(MAIL, "| /usr/sbin/sendmail -t") or die("Ошибка $!");
print MAIL $header;
print MAIL $msg;
close(MAIL) or die("Ошибка $!");

print "Письмо отправлено";
```

Листинг 3.40. Пример письма в формате HTML

```
my $header = "From: support <support\@site.ru>\n";
$header .= "To: Ivan <ivan\@mail.ru>\n";
$header .= "Subject: Test mail\n";
$header .= "Content-Type: text/html; charset=windows-1251\n";
$header .= "Content-Transfer-Encoding: 8bit\n\n";

my $msg = "Добрый день!<BR><BR>\n";
$msg .= "Вы успешно зарегистрированы.<BR><BR>\n";
```

```
$msg .= "<A href=\"http://www.site.ru/\">http://www.site.ru/</A><BR>\n";
$msg .= "support\@site.ru";

open(MAIL, "| /usr/sbin/sendmail -t") or die("Ошибка $!");
print MAIL $header;
print MAIL $msg;
close(MAIL) or die("Ошибка $!");

print "Письмо отправлено";
```

Обратите внимание

Символ @ является специальным. По этой причине его необходимо экранировать с помощью защитного слэша.

Теперь рассмотрим возможность рассылки писем по E-mail-адресам из файла (листинг 3.41). E-mail-адреса должны располагаться по одному на строке. Например:

```
info@site.ru
admin@site.ru
```

Листинг 3.41. Рассылка писем

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use Fcntl qw( :DEFAULT :flock );
print "Content-type: text/html\n\n";

print "<HTML><HEAD>\n";
print "<TITLE>Рассылка писем</TITLE>\n";
print "</HEAD>\n";
print "<BODY>\n";

if (defined(param("send"))) {
   my ($header, $msg);
   my $pattern = qr/^([a-z0-9_\.\-]+)\@(([a-z0-9\-]+\.)+[a-z]{2,4})$/;
   $msg = "Добрый день!\n\n";
   $msg .= "Новости нашего сайта.\n\n";
   $msg .= "http://www.site.ru/\n";
   $msg .= "mail\@site.ru";
```

```
   sysopen(FILE, 'mail.txt', O_RDONLY) or die("Ошибка $!");
   flock(FILE, LOCK_EX) or die("Ошибка $!");
   while (<FILE>) {
      chomp($_);
      if ($_ =~ m/$pattern/i) {
         $header = "From: support <support\@site.ru>\n";
         $header .= "To: $_\n";
         $header .= "Subject: News\n";
         $header .= "Content-Type: text/plain; charset=windows-1251\n";
         $header .= "Content-Transfer-Encoding: 8bit\n\n";
         open(MAIL, "| /usr/sbin/sendmail -t") or die("Ошибка $!");
         print MAIL $header;
         print MAIL $msg;
         close(MAIL) or die("Ошибка $!");
         print "$_ — OK<BR>\n";
      }
      else {
         print "$_ — Ошибка<BR>\n";
      }
   }
   flock(FILE, LOCK_UN) or die("Ошибка $!");
   close(FILE);
   print "Сообщения разосланы";
}
else {
   print '<FORM action="http://perlbook.ru/cgi-bin/mail.pl">', "\n";
   print '<INPUT type="hidden" name="send" value="1">', "\n";
   print '<INPUT type="submit" value="Разослать">', "\n";
   print "</FORM>\n";
}
print "</BODY>\n";
print "</HTML>\n";
```

При нажатии на кнопку **Разослать** на все E-mail из файла будет отправлено письмо.

Обратите внимание, на локальной машине письма отправлены не будут, мы не устанавливали программу отправки писем — `Sendmail`. Тем не менее, в состав Денвера входит эмулятор программы `Sendmail`. В этом случае письма

не отправляются адресату, а сохраняются в папке C:\WebServers\tmp \!sendmail в виде файлов. Просмотреть содержимое файла позволяет любой текстовый редактор, а также программа Outlook Express. На сервере хостинг-провайдера программа `Sendmail` практически всегда установлена и настроена. Правда, количество одновременно отправленных писем часто ограничено.

При рассылке в письме обязательно должна быть предусмотрена возможность отписаться от рассылки. И запомните: рассылка спама в Интернете запрещена. Под спамом понимаются письма, не запрошенные явным способом получателем.

3.7. Создание изображений с помощью модуля GD

Модуль `GD` позволяет создать новое изображение, использовать готовое изображение в качестве основы, изменить размер, выводить текст в изображение и многое другое. Позволяет работать с форматами JPEG, PNG, GIF и некоторыми другими. Модуль широко используется для создания счетчиков посещений, графиков, демонстрации уменьшенных копий изображений и управлении показами баннеров.

Модуль не входит в состав ActivePerl, и нет возможности установить его с сайта **http://www.activestate.com/**. Для установки модуля `GD` запускаем Денвер. Переходим на диск Z и запускаем файл Z:\usr\local\perl\bin\ppm-shell.bat. В командной строке должно быть приглашение:

```
ppm>
```

Набираем команду:

```
install http://theoryx5.uwinnipeg.ca/ppms/GD.ppd
```

Далее нажимаем клавишу <Enter>.

> ***Обратите внимание***
>
> Файл ppm-shell.bat необходимо обязательно запустить с диска Z. В противном случае модуль будет установлен в каталог C:\usr, а не в C:\WebServers\usr.

3.7.1. Получение информации об изображении

Для получения размеров изображения используются следующие методы:

- `getBounds()` — возвращает длину и ширину изображения в виде списка:

```
my ($width, $height) = $img->getBounds();
print "Ширина " . $width . "<BR>\n";
print "Высота " . $height . "<BR>\n";
```

- `width()` — возвращает ширину изображения:

```
print "Ширина " . $img->width() . "<BR>\n";
```

- `height()` — возвращает высоту изображения:

```
print "Высота " . $img->height() . "<BR>\n";
```

Чтобы определить тип изображения, можно воспользоваться функцией `_image_type()` из модуля `GD::Image`. Функция возвращает пустую строку в случае ошибки или одно из следующих значений:

- `Png`;
- `Jpeg`;
- `Gif`;
- `Gd2`;
- `Xpm`.

Функция `_image_type()` может быть полезна для определения типа изображения при загрузке его с помощью формы (листинг 3.42).

Листинг 3.42. Определение типа изображения

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
print "Content-type: text/html\n\n";

use GD;
my $myImage;
my $file = param('file_name');
# Выделяем имя файла
$file =~ m/([^\/\\:]+)$/;
my $name_file = $1;
{
local $/ = "";
$myImage = <$file>;
}
print "Имя файла: " . $name_file . "<BR>";
my $type = GD::Image::_image_type($myImage);
if ($type eq 'Gif' || $type eq 'Jpeg' || $type eq 'Png') {
   print "Тип изображения: $type";
}
```

```
else {
   print "Недопустимый тип файла";
}
```

3.7.2. Создание холста

Новое изображение создается с помощью метода `new()`:

```
use GD;
<Идентификатор> = new GD::Image([<Ширина>, <Высота>]);
```

Для загрузки готового изображения в качестве холста используются следующие методы:

- `<Идентификатор> = GD::Image->newFromPng(<Имя файла>)` — для изображений в формате PNG;
- `<Идентификатор> = GD::Image->newFromGif(<Имя файла>)` — для изображений в формате GIF;
- `<Идентификатор> = GD::Image->newFromJpeg(<Имя файла>)` — для изображений в формате JPEG.

Вся дальнейшая работа с изображением осуществляется через созданный идентификатор (дескриптор).

3.7.3. Вывод созданного изображения

Чтобы вывести изображение в Web-браузер, нужно вначале вывести заголовок, соответствующий формату изображения:

```
print "Content-type: image/png\n\n";
print "Content-type: image/gif\n\n";
print "Content-type: image/jpeg\n\n";
```

А затем следует вывести изображение с помощью соответствующего формату метода:

- `png()` — для изображений в формате PNG:

  ```
  binmode(STDOUT);
  print <Идентификатор>->png([<Сжатие>]);
  ```

  `<Сжатие>` — число от 0 до 9;
- `gif()` — для изображений в формате GIF:

  ```
  binmode(STDOUT);
  print <Идентификатор>->gif();
  ```

- `jpeg()` — для изображений в формате JPEG:

```
binmode(STDOUT);
print <Идентификатор>->jpeg([<Сжатие>]);
```

`<Сжатие>` — число от 0 до 100.

Изображение можно вывести не только в Web-браузер, но и сохранить в файл:

```
use GD;
my $img = new GD::Image(100, 100);
open (IMG, ">img.png");
binmode(IMG);
print IMG $img->png();
close(IMG);
```

В качестве примера выведем баннер banner.gif в окно Web-браузера. Для этого создадим файл banner.pl в папке C:\WebServers\home\perlbook.ru\cgi-bin с кодом из листинга 3.43.

Листинг 3.43. Вывод изображения в окно Web-браузера

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: image/gif\n\n";

use GD;
my $path = "C:\\WebServers\\home\\perlbook.ru\\www\\banner.gif";
my $img = GD::Image->newFromGif($path);
binmode(STDOUT);
print $img->gif();
```

Вывести баннер в окно Web-браузера в любом документе позволяет следующий HTML-код:

```
<IMG src="http://perlbook.ru/cgi-bin/banner.pl">
```

Это аналогично встраиванию обычного изображения:

```
<IMG src="http://perlbook.ru/banner.gif">
```

Обратите внимание

Во время записи в файл или вывода изображения в Web-браузер необходимо включить двоичный режим с помощью функции `binmode()`.

3.7.4. Работа с цветом

Добавить новый цвет в палитру позволяет метод `colorAllocate()`:

```
<Идентификатор>->colorAllocate(<Доля красного>, <Доля зеленого>,
<Доля синего>);
```

Например:

```
use GD;
my $img = new GD::Image(100, 100);
my $red = $img->colorAllocate(255, 0, 0);
```

Следует отметить, что первый цвет, добавляемый в палитру, становится цветом фона. По этой причине код из листинга 3.44 выведет красный квадрат:

Листинг 3.44. Задание цвета фона

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: image/gif\n\n";

use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $red = $img->colorAllocate(255, 0, 0);
my $white = $img->colorAllocate(255, 255, 255);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Метод `colorDeallocate()` удаляет из палитры цвет, который был создан ранее методом `colorAllocate()`:

```
<Идентификатор>->colorDeallocate(<Цвет>);
```

Например:

```
use GD;
my $img = new GD::Image(100, 100);
my $red = $img->colorAllocate(255, 0, 0);
my $white = $img->colorAllocate(255, 255, 255);
$img->colorDeallocate($white);
```

С помощью метода `transparent()` можно сделать определенный цвет из палитры прозрачным:

```
<Идентификатор>->transparent(<Цвет>);
```

В качестве примера сделаем прозрачным цвет фона и отобразим на изображении контур круга черным цветом (листинг 3.45).

Листинг 3.45. Задание прозрачного цвета фона

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: image/gif\n\n";

use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
$img->arc(50, 50, 80, 80, 0, 360, $black);
$img->transparent($white);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Сохраняем скрипт под названием banner.pl. Далее выводим изображение на Web-странице. Для демонстрации эффекта сделаем фон Web-страницы красного цвета:

```
<HTML><HEAD>
<TITLE>Изображения</TITLE>
</HEAD>
<BODY bgcolor="#FF0000">
<IMG src="http://perlbook.ru/cgi-bin/banner.pl">
</BODY></HTML>
```

В итоге отобразится контур круга на красном фоне.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> При использовании метода `jpeg()` эффекта не будет.

Метод `getPixel()` возвращает цвет указанной точки изображения:

```
$color = <Идентификатор>->getPixel(<X>, <Y>);
```

Чтобы получить числовое значение цвета, можно воспользоваться методом `rgb()`:

```
($red,$green,$blue) = <Идентификатор>->rgb(<Цвет>);
```

Для примера выведем желтый квадрат и получим цвет пиксела внутри квадрата, а затем выведем его числовое значение (листинг 3.46).

Листинг 3.46. Определение цвета пиксела

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: text/html\n\n";

use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $yellow = $img->colorAllocate(255, 255, 0);
$img->filledRectangle(5, 5, 80, 80, $yellow);
my $newcolor = $img->getPixel(10, 10);
my ($r, $g, $b) = $img->rgb($newcolor);
print $r, " — ", $g, " — ", $b;
```

Вывод:

```
255 — 255 — 0
```

Метод `colorsTotal()` возвращает количество цветов в палитре изображения:

```
$count = <Идентификатор>->colorsTotal();
```

Метод `fill()` позволяет закрасить область одного цвета другим цветом. Достаточно указать координаты точки и нужный цвет:

```
<Идентификатор>->fill(<X>, <Y>, <Цвет>);
```

Метод `fillToBorder()` позволяет закрасить область разного цвета другим цветом. Достаточно указать координаты точки, цвет границы и нужный цвет:

```
<Идентификатор>->fillToBorder(<X>, <Y>, <Цвет границы>, <Цвет>);
```

В качестве примера создадим контур квадрата черного цвета, а внутри квадрата контур круга желтого цвета. Далее закрасим область квадрата красным цветом, указав точку внутри круга (листинг 3.47).

Листинг 3.47. Метод `fillToBorder()`

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: image/gif\n\n";
```

```
use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $yellow = $img->colorAllocate(255, 255, 0);
my $black = $img->colorAllocate(0, 0, 0);
my $red = $img->colorAllocate(255, 0, 0);
$img->rectangle(5, 5, 95, 95, $black);
$img->arc(50, 50, 20, 20, 0, 360, $yellow);
$img->fillToBorder(50, 50, $black, $red);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Допустим, вместо метода `fillToBorder()` можно было указать метод `fill()`:

```
$img->fill(50, 50, $red);
```

Тогда мы получили бы только закрашенный круг, а не квадрат.

Метод `colorClosest()` возвращает ближайший к указанному цвет из имеющихся в палитре. При неудаче возвращает значение `-1`:

```
$color = <Идентификатор>->colorClosest(<Доля красного>, <Доля зеленого>,
<Доля синего>);
```

Например:

```
use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $yellow = $img->colorAllocate(255, 255, 0);
my $color = $img->colorClosest(255, 255, 1);
my ($r, $g, $b) = $img->rgb($color);
print $r, " — ", $g, " — ", $b;
```

Вывод:

```
255 — 255 — 0
```

3.7.5. Рисование линий и фигур

Модуль `GD` позволяет рисовать следующие фигуры:

- точку:

```
<Идентификатор>->setPixel(<X>, <Y>, <Цвет>);
```

<X> и <Y> — координаты точки:

```
$img->setPixel(50, 50, $red);
```

- сплошную линию:

```
<Идентификатор>->line(<X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

<X1> и <Y1> — координаты первой точки, <X2> и <Y2> — координаты второй точки:

```
$img->line(20, 50, 70, 50, $red);
```

- пунктирную линию:

```
<Идентификатор>->dashedLine(<X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

<X1> и <Y1> — координаты первой точки, <X2> и <Y2> — координаты второй точки:

```
$img->dashedLine(20, 50, 70, 50, $red);
```

- прямоугольник без заливки:

```
<Идентификатор>->rectangle(<X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

<X1> и <Y1> — координаты левого верхнего угла, <X2> и <Y2> — координаты правого нижнего угла, <Цвет> — цвет границы:

```
$img->rectangle(5, 5, 95, 95, $red);
```

- прямоугольник с заливкой:

```
<Идентификатор>->filledRectangle(<X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

<X1> и <Y1> — координаты левого верхнего угла, <X2> и <Y2> — координаты правого нижнего угла, <Цвет> — цвет прямоугольника:

```
$img->filledRectangle(5, 5, 95, 95, $red);
```

- эллипс без заливки:

```
<Идентификатор>->ellipse(<X>, <Y>, <Ширина>, <Высота>, <Цвет>);
```

<X> и <Y> — координаты центра, <Ширина> — ширина, <Высота> — высота, <Цвет> — цвет границы:

```
$img->ellipse(50, 50, 60, 30, $red);
```

- эллипс с заливкой:

```
<Идентификатор>->filledEllipse(<X>, <Y>, <Ширина>, <Высота>, <Цвет>);
```

<X> и <Y> — координаты центра, <Ширина> — ширина, <Высота> — высота, <Цвет> — цвет эллипса:

```
$img->filledEllipse(50, 50, 60, 30, $red);
```

- дугу, круг, эллипс без заливки:

```
<Идентификатор>->arc(<X>, <Y>, <Ширина>, <Высота>, <Старт>, <Конец>,
<Цвет>);
```

<X> и <Y> — координаты центра, <Ширина> — ширина, <Высота> — высота, <Старт> — начальный угол, <Конец> — конечный угол, <Цвет> — цвет границы:

```
$img->arc(50, 50, 50, 50, 0, 360, $red);
```

- дугу, круг, эллипс с заливкой:

```
<Идентификатор>->filledArc(<X>, <Y>, <Ширина>, <Высота>, <Старт>, <Конец>, <Цвет>, [<Стиль>]);
```

<X> и <Y> — координаты центра, <Ширина> — ширина, <Высота> — высота.

<Старт> — начальный угол, <Конец> — конечный угол, <Цвет> — цвет фигуры, <Стиль> — стиль заливки.

Необязательный параметр <Стиль> может принимать следующие значения (или их комбинацию):

- gdArc — выводится контур, соединяющий начальную и конечную точки по часовой стрелке. Заливка производится без сектора, соединяющего центр фигуры с начальной и конечной точками;
- gdChord — выводится прямая линия, соединяющая начальную и конечную точки. Закрашивается часть фигуры, которая была бы не закрашена при указании значения gdArc;
- gdNoFill — фигура без заливки. Отображается только контур;
- gdEdged — выводится полностью контур фигуры с заливкой. Если начальная точка не совпадает с конечной, то они соединяются через центр фигуры.

В качестве примера выведем результат применения перечисленных значений (листинг 3.48).

Листинг 3.48. Метод `filledArc()`

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
print "Content-type: image/gif\n\n";

use GD;
my $img = new GD::Image(300, 80);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
```

```
$img->rectangle(0, 0, 299, 79, $black);
$img->filledArc(25, 40, 40, 40, 30, 360, $black);
$img->filledArc(75, 40, 40, 40, 30, 360, $black, gdArc);
$img->filledArc(125, 40, 40, 40, 30, 360, $black, gdChord);
$img->filledArc(175, 40, 40, 40, 30, 360, $black, gdNoFill);
$img->filledArc(225, 40, 40, 40, 30, 360, $black, gdEdged);
$img->filledArc(275, 40, 40, 40, 30, 360, $black, gdEdged | gdNoFill);

# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Метод `setThickness()` устанавливает толщину линий при рисовании. По умолчанию толщина линий составляет 1 пиксел:

```
<Идентификатор>->setThickness(<Толщина в пикселах>);
```

Например:

```
$img->setThickness(5);
$img->line(20, 50, 70, 50, $red);
```

3.7.6. Рисование многоугольников

Для рисования многоугольников необходимо с помощью экземпляра класса `GD::Polygon` описать координаты вершин. Класс предоставляет следующие методы:

- `new()` — создает новый экземпляр класса `GD::Polygon`:

  ```
  my $polygon = GD::Polygon->new();
  ```

- `addPt(<X>, <Y>)` — создает новую вершину:

  ```
  my $polygon = GD::Polygon->new();
  $polygon->addPt(20, 20);
  $polygon->addPt(20, 70);
  $polygon->addPt(80, 70);
  ```

- `setPt(<Индекс вершины>, <X>, <Y>)` — изменяет координаты указанной вершины. Первая вершина имеет индекс 0:

  ```
  my $polygon = GD::Polygon->new();
  $polygon->addPt(20, 20);
  $polygon->addPt(20, 70);
  $polygon->addPt(80, 70);
  $polygon->setPt(0, 25, 20);
  ```

- `getPt(<Индекс вершины>)` — возвращает координаты указанной вершины:

```
use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $red = $img->colorAllocate(255, 0, 0);

my $polygon = GD::Polygon->new();
$polygon->addPt(20, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
my($x, $y) = $polygon->getPt(0);
print $x, " — ", $y;
```

Вывод:

```
20 — 20
```

- `deletePt(<Индекс вершины>)` — удаляет указанную вершину:

```
$polygon->deletePt(1);
```

- `clear()` — удаляет все вершины:

```
$polygon->clear();
```

- `length()` — возвращает количество вершин:

```
my $polygon = GD::Polygon->new();
$polygon->addPt(20, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
print $polygon->length();
```

Вывод:

```
3
```

- `vertices()` — возвращает координаты всех вершин:

```
my $polygon = GD::Polygon->new();
$polygon->addPt(20, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
my @vertices = $polygon->vertices();
foreach (@vertices) {
   print join(", ", @$_), "<BR>";
}
```

Вывод:

```
20, 20
20, 70
80, 70
```

- `bounds()` — возвращает координаты прямоугольника, в который вписан многоугольник:

```
my $polygon = GD::Polygon->new();
$polygon->addPt(30, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
my @bounds = $polygon->bounds();
print $bounds[0], " — ", $bounds[1], "<BR>";
print $bounds[2], " — ", $bounds[3];
```

Вывод:

```
20 — 20
80 — 70
```

Вывести многоугольник позволяют следующие методы класса `GD::Image`:

- `polygon()` — многоугольник без заливки:

```
<Идентификатор>->polygon(<Экземпляр класса GD::Polygon>, <Цвет>);
```

`<Цвет>` — цвет границы. Пример:

```
use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $red = $img->colorAllocate(255, 0, 0);

my $polygon = GD::Polygon->new();
$polygon->addPt(20, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
$img->polygon($polygon, $red);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

- `filledPolygon()` — многоугольник с заливкой:

```
<Идентификатор>->filledPolygon(<Экземпляр класса GD::Polygon>, <Цвет>);
```

<Цвет> — цвет многоугольника. Пример:

```
use GD;
my $img = new GD::Image(100, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $red = $img->colorAllocate(255, 0, 0);

my $polygon = GD::Polygon->new();
$polygon->addPt(20, 20);
$polygon->addPt(20, 70);
$polygon->addPt(80, 70);
$img->filledPolygon($polygon, $red);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

3.7.7. Вывод текста в изображение. Создаем счетчик посещений

Для вывода текста используются следующие функции:

- `char()` — рисует символ на изображении по горизонтали:

```
<Идентификатор>->char(<Шрифт>, <X>, <Y>, <Символ>, <Цвет>);
$img->char(gdGiantFont, 5, 40, "S", $black);
```

- `charUp()` — рисует символ на изображении по вертикали:

```
<Идентификатор>->charUp(<Шрифт>, <X>, <Y>, <Символ>, <Цвет>);
$img->charUp(gdGiantFont, 5, 40, "T", $black);
```

- `string()` — рисует строку на изображении по горизонтали:

```
<Идентификатор>->string(<Шрифт>, <X>, <Y>, <Строка>, <Цвет>);
$img->string(gdGiantFont, 5, 40, "String", $black);
```

- `stringUp()` — рисует строку на изображении по вертикали:

```
<Идентификатор>->stringUp(<Шрифт>, <X>, <Y>, <Строка>, <Цвет>);
$img->stringUp(gdGiantFont, 5, 70, "String", $black);
```

Параметр <Шрифт> в методах `char()`, `charUp()`, `string()` и `stringUp()` может принимать значения `gdGiantFont`, `gdLargeFont`, `gdMediumBoldFont`, `gdSmallFont` и `gdTinyFont`. Для сравнения размеров шрифтов выведем пять строк разными шрифтами (листинг 3.49).

Листинг 3.49. Сравнение размеров шрифтов

```
use GD;
my $img = new GD::Image(120, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
$img->rectangle(0, 0, 119, 99, $black);

$img->string(gdGiantFont, 5, 10, "gdGiantFont", $black);
$img->string(gdLargeFont, 5, 30, "gdLargeFont", $black);
$img->string(gdMediumBoldFont, 5, 50, "gdMediumBoldFont", $black);
$img->string(gdSmallFont, 5, 65, "gdSmallFont", $black);
$img->string(gdTinyFont, 5, 85, "gdTinyFont", $black);

# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Для получения размеров шрифта можно воспользоваться методами `width()` и `height()` (листинг 3.50).

Листинг 3.50. Методы `width()` и `height()`

```
use GD;
print gdGiantFont->width(), "<BR>";
print gdGiantFont->height(), "<BR>";
```

Вывод:

```
9
15
```

Все эти методы с буквами русского языка не работают. Для русского языка следует использовать TrueType-шрифты (например, arial.ttf). В Windows XP шрифты расположены в папке C:\WINDOWS\Fonts.

Метод `stringFT()` выводит строку на изображении TrueType-шрифтом:

```
<Идентификатор>->stringFT(<Цвет>, <Шрифт>, <Размер>, <Угол>, <X>, <Y>,
<Строка>);
```

`<Цвет>` — цвет шрифта, `<Шрифт>` — имя файла со шрифтом, `<Размер>` — размер шрифта, `<Угол>` — угол поворота текста в радианах. Поворот осуществ-

ляется против часовой стрелки. Для преобразования градусов в радианы можно воспользоваться функцией `deg2rad()` из модуля `Math::Trig`:

```
use Math::Trig;
$rad = deg2rad(90);
```

<X> и <Y> — координаты левой крайней точки базовой линии, <Строка> — строка для вывода. Строку на русском языке необходимо перекодировать в кодировку `UTF-8`. Например, с помощью модуля `Text::Iconv`.

В качестве примера выведем строку на русском языке под углом 45° (листинг 3.51).

Листинг 3.51. Вывод строки под углом

```
use GD;
use Math::Trig;
use Text::Iconv;
my $img = new GD::Image(120, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
$img->rectangle(0, 0, 119, 99, $black);
my $fontfile = "C:/WINDOWS/Fonts/arial.ttf";
# Преобразуем кодировку
my $iconv1 = Text::Iconv->new("windows-1251", "UTF-8");
my $str = $iconv1->convert("Строка");
# Выводим строку на русском языке под углом 45°
$img->stringFT($black, $fontfile, 20, deg2rad(45), 30, 80, $str);
# Вывод изображения
binmode(STDOUT);
print $img->gif();
```

Метод `stringFT()` возвращает массив координат 4 вершин прямоугольника, в который будет вписан текст (листинг 3.52). Вершины перечисляются в следующем порядке: нижняя левая, нижняя правая, верхняя правая, верхняя левая.

Листинг 3.52. Метод `stringFT()`

```
use GD;
my $img = new GD::Image(100, 40);
```

```
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
my $fontfile = "C:/WINDOWS/Fonts/arial.ttf";
# Выводим строку
my @bounds = $img->stringFT($black, $fontfile, 20, 0, 11, 21, "Test");
print $bounds[0], " — ", $bounds[1], "<BR>";
print $bounds[2], " — ", $bounds[3], "<BR>";
print $bounds[4], " — ", $bounds[5], "<BR>";
print $bounds[6], " — ", $bounds[7], "<BR>";
```

Вывод:

```
9 — 22 — координаты левого нижнего угла.
64 — 22 — координаты правого нижнего угла.
64 — 0 — координаты правого верхнего угла.
9 — 0 — координаты левого верхнего угла.
```

Если необходимо получить только координаты вершин прямоугольника, то можно воспользоваться методом `GD::Image->stringFT()`:

```
<Массив> = GD::Image->stringFT(<Цвет>, <Шрифт>, <Размер>, <Угол>, <X>,
<Y>, <Строка>);
```

Метод `GD::Image->stringFT()` возвращает массив координат 4 вершин прямоугольника, в который будет вписан текст. Вершины перечисляются в следующем порядке: нижняя левая, нижняя правая, верхняя правая, верхняя левая. В качестве примера произведем выравнивание выводимого текста по центру, как по вертикали, так и по горизонтали (листинг 3.53).

Листинг 3.53. Метод `GD::Image->stringFT()`

```
use GD;
use Text::Iconv;
my $img = new GD::Image(120, 100);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $black = $img->colorAllocate(0, 0, 0);
$img->rectangle(0, 0, 119, 99, $black);
my $fontfile = "C:/WINDOWS/Fonts/arial.ttf";
# Преобразуем кодировку
my $iconv1 = Text::Iconv->new("windows-1251", "UTF-8");
```

```
my $str = $iconv1->convert("Строка");
my @bounds = GD::Image->stringFT($black, $fontfile, 20, 0, 0, 0, $str);
my $w = int(($img->width() — $bounds[4] — $bounds[6])/2);
my $h = int(($img->height() — $bounds[1] — $bounds[7])/2);
# Выводим строку на русском языке
$img->stringFT($black, $fontfile, 20, 0, $w, $h, $str);
# Выводим изображение
binmode(STDOUT);
print $img->gif();
```

ОБРАТИТЕ ВНИМАНИЕ

Если текст повернут под углом, то вычисление местоположения при центрировании будет другим.

Рассмотрим еще один пример. Создадим счетчик посещения с использованием cookies и выведем результат в графическом виде (листинг 3.54).

Листинг 3.54. Счетчик посещений

```
use CGI qw( :standard);
my $id_count;
if (!cookie('id_count')) {
   $id_count = 0;
}
else {
   $id_count = cookie('id_count');
}
$id_count++;
my $cookies = cookie(-name=>'id_count', -value=>$id_count,
-expires=>'+360d', -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies\n";
print "Content-type: image/gif\n\n";

use GD;
use Text::Iconv;
my $img = new GD::Image(88, 31);
# Добавляем цвета в палитру
my $white = $img->colorAllocate(255, 255, 255);
my $grey = $img->colorAllocate(128, 128, 128);
```

```
my $black = $img->colorAllocate(0, 0, 0);
$img->rectangle(0, 0, 87, 30, $black);
my $fontfile = "C:/WINDOWS/Fonts/arial.ttf";
# Преобразуем кодировку
my $iconv1 = Text::Iconv->new("windows-1251", "UTF-8");
my $str = $iconv1->convert("Мой счетчик");
$img->stringFT($grey, $fontfile, 8, 0, 12, 13, $str);
my @bounds = GD::Image->stringFT($black, $fontfile, 12, 0, 0, 0,
$id_count);
my $w = int(($img->width() - $bounds[4] - $bounds[6])/2);
# Выводим значение счетчика
$img->stringFT($black, $fontfile, 12, 0, $w, 27, $id_count);
# Выводим изображение
binmode(STDOUT);
print $img->gif();
```

Попробуйте обновить страницу, цифры на счетчике будут увеличиваться.

3.7.8. Преобразование изображений

Метод `copy()` позволяет копировать части изображения. Имеет следующий формат:

```
<image1>->copy(<image2>, <X1>, <Y1>, <X2>, <Y2>, <Ширина>, <Высота>)
```

`<image1>` — изображение, в которое добавляем фрагмент, `<image2>` — изображение, из которого копируем фрагмент, `<X1>` и `<Y1>` — координаты точки для вставки фрагмента, `<X2>` и `<Y2>` — координаты начальной точки копируемого фрагмента, `<Ширина>` и `<Высота>` — ширина и высота копируемого фрагмента.

Для примера создадим два изображения. Первое — красного цвета. На втором изображении зеленого цвета выведем черный квадрат. Теперь скопируем этот квадрат и по одному пикселу по сторонам квадрата, а затем вставим фрагмент в первое изображение, начиная с точки с координатами 2, 2 (листинг 3.55).

Листинг 3.55. Метод `copy()`

```
use GD;
my $img1 = new GD::Image(100, 100);
my $red = $img1->colorAllocate(255, 0, 0);
```

```
my $img2 = new GD::Image(100, 100);
my $green = $img2->colorAllocate(0, 255, 0);
my $black = $img2->colorAllocate(0, 0, 0);
$img2->filledRectangle(10, 10, 50, 50, $black);
$img1->copy($img2, 2, 2, 9, 9, 43, 43);


# Вывод изображения
binmode(STDOUT);
print $img1->gif();
```

Метод `copyResized()` позволяет копировать и изменять размеры части изображения. Имеет следующий формат:

```
<image1>->copyResized(<image2>, <X1>, <Y1>, <X2>, <Y2>, <W1>, <H1>, <W2>,
<H2>)
```

`<image1>` — изображение, в которое добавляем фрагмент, `<image2>` — изображение, из которого копируем фрагмент, `<X1>` и `<Y1>` — координаты точки для вставки фрагмента, `<X2>` и `<Y2>` — координаты начальной точки копируемого фрагмента, `<W1>` и `<H1>` — новая ширина и высота фрагмента, `<W2>` и `<H2>` — исходная ширина и высота фрагмента.

Для примера рассмотрим уменьшение изображения в два раза (листинг 3.56).

Листинг 3.56. Метод `copyResized()`

```
use GD;
my $path = "C:/WebServers/home/perlbook.ru/www/banner.gif";
my $img1 = GD::Image->newFromGif($path);
my $w_old = $img1->width();
my $h_old = $img1->height();
my $w_new = $img1->width()/2;
my $h_new = $img1->height()/2;
my $img2 = new GD::Image($w_new, $h_new);
my $white = $img2->colorAllocate(255, 255, 255);
$img2->transparent($white);
$img2->copyResized($img1, 0, 0, 0, 0, $w_new, $h_new, $w_old, $h_old);
# Вывод изображения
binmode(STDOUT);
print $img2->gif();
```

Помимо методов `copy()` и `copyResized()` для преобразования изображений используются следующие методы:

- `rotate180()` — поворачивает изображение на 180°:

```
#!/usr/bin/perl -w
print "Content-type: image/gif\n\n";
use GD;
my $img1 = GD::Image->newFromGif("../www/banner.gif");
$img1->rotate180();
binmode(STDOUT);
print $img1->gif();
```

- `flipHorizontal()` — создает горизонтальный зеркальный образ изображения:

```
$img1->flipHorizontal();
```

- `flipVertical()` — создает вертикальный зеркальный образ изображения:

```
$img1->flipVertical();
```

- `copyRotate90()` — копирует изображение и поворачивает его на 90°:

```
my $img2 = $img1->copyRotate90();
```

- `copyRotate180()` — копирует изображение и поворачивает его на 180°:

```
my $img2 = $img1->copyRotate180();
```

- `copyRotate270()` — копирует изображение и поворачивает его на 270°:

```
my $img2 = $img1->copyRotate270();
```

- `copyFlipHorizontal()` — копирует изображение и создает горизонтальный зеркальный образ:

```
my $img2 = $img1->copyFlipHorizontal();
```

- `copyFlipVertical()` — копирует изображение и создает вертикальный зеркальный образ:

```
my $img2 = $img1->copyFlipVertical();
```

3.8. Работа с базами данных MySQL и Access

На сегодняшний день ни один крупный портал не обходится без использования систем управления базами данных (СУБД). В Perl существует единый

интерфейс для всех баз данных — DBI (Data Base Interface). Для доступа к конкретной базе данных используется соответствующий драйвер DBD. Например, для доступа к MySQL используется драйвер DBD::mysql. DBI и DBD обеспечивают эффективную поддержку баз данных и позволяют работать с MySQL, Oracle, Sybase, Informix, SQLite и др. С помощью драйвера DBD::ODBC мы можем подключиться к любой базе данных, поддерживающей драйвер ODBC, например, к базе данных Access.

В этой главе мы рассмотрим технологию доступа к базам данных MySQL и Access. Для доступа к этим базам данных необходимо установить модули DBD::mysql и DBD::ODBC. Проверить, установлены ли модули, можно с помощью метода available_drivers():

```
#!/usr/bin/perl -w
use DBI;
print "Content-type: text/html\n\n";

my @drivers = DBI->available_drivers();
print join("<BR>\n", @drivers);
```

Вывод:

```
DBM
ExampleP
File
Proxy
SQLite
Sponge
mysql
```

Драйвер mysql будет доступен, если были выполнены инструкции *разд. 1.1.3*. Драйвер ODBC также по умолчанию не установлен. Если необходимых драйверов в списке нет, то необходимо их установить. Запускаем Денвер и подключаемся к Интернету. Переходим на диск Z и запускаем файл Z:\usr\local\perl\bin\ppm-shell.bat. В командной строке должно быть приглашение:

```
ppm>
```

Для установки драйвера mysql вводим команду:

```
install http://theoryx5.uwinnipeg.ca/ppms/DBD-mysql.ppd
```

Для установки драйвера ODBC вводим команду:

```
install DBD-ODBC
```

Обратите внимание

Файл ppm-shell.bat должен быть обязательно запущен с диска Z. В противном случае модули будут установлены в каталог C:\usr, а не в C:\WebServers\usr.

С помощью метода `data_sources()` можно получить список всех баз данных, зарегистрированных в системе, для указанного в качестве параметра драйвера. Например, мы можем посмотреть, к каким базам данных можно подключиться с помощью драйвера `ODBC`:

```
use DBI;
my @sources = DBI->data_sources("ODBC");
print join("<BR>\n", @sources);
```

Вывод:

```
dbi:ODBC:Visual FoxPro Tables
dbi:ODBC:Visual FoxPro Database
dbi:ODBC:База данных MS Access
dbi:ODBC:Файлы dBASE
dbi:ODBC:Файлы Excel
dbi:ODBC:База данных Visual FoxPro
dbi:ODBC:Таблицы Visual FoxPro
dbi:ODBC:Файлы dBase — Word
dbi:ODBC:Файлы FoxPro — Word
```

Как видно из примера, мы можем подключиться не только к базе данных Access, но и к FoxPro, dBASE и файлам Excel.

Теперь создадим базу данных MySQL и таблицу в ней для дальнейшего рассмотрения примеров. Для этого можно воспользоваться программой phpMyAdmin. Запускаем Денвер и открываем Web-браузер. В адресной строке набираем:

```
http://localhost/Tools/phpMyAdmin/
```

В поле **Создать новую БД** вводим `tests`. Из списка **Сравнение** выбираем пункт **cp1251_general_ci**. Нажимаем кнопку **Создать**. Отобразится сообщение: **БД tests была создана**. Для создания таблицы переходим на вкладку **SQL**. В поле **Выполнить SQL запрос(ы) на БД** вводим SQL-команду:

```
CREATE TABLE city (
   id_City INT NOT NULL auto_increment,
   City CHAR(50) NOT NULL,
   PRIMARY KEY (id_City)
) ENGINE = MYISAM DEFAULT CHARSET = cp1251;
```

Нажимаем кнопку **Пошел**. Название новой таблицы отобразится в левой части окна Web-браузера в виде ссылки. Если перейти по этой ссылке, то в правой части отобразится структура таблицы. Теперь добавим две записи. Для этого на вкладке SQL вводим команды:

```
INSERT INTO city VALUES(NULL, 'Санкт-Петербург');
INSERT INTO city VALUES(NULL, 'Москва');
```

Теперь можно приступать к изучению интерфейса баз данных Perl.

3.8.1. Установка соединения

Для установки соединения используется метод `connect()`:

```
<Идентификатор> = DBI->connect(<Строка для подключения>,
<Имя пользователя>, <Пароль>, [<Атрибуты>]);
```

Метод возвращает идентификатор соединения. Вся дальнейшая работа с базой данных осуществляется через этот идентификатор. В случае неудачи возвращается значение `undef`, а в переменной `$DBI::errstr` сохраняется сообщение об ошибке. Номер ошибки сохраняется в переменной `$DBI::err`.

Параметр `<Строка для подключения>` имеет следующий формат:

```
DBI:<Драйвер>:<Имя базы данных>:<Хост>:<Порт>
```

В необязательном параметре `<Атрибуты>` могут быть указаны следующие атрибуты:

- `PrintError` — при установленном атрибуте выводятся все предупредительные сообщения (включен по умолчанию);
- `RaiseError` — при установленном атрибуте все ошибки приводят к прерыванию выполнения программы (по умолчанию выключен);
- `AutoCommit` — оказывает влияние на поддержку транзакций (включен по умолчанию).

Для сброса значения атрибуту необходимо присвоить значение 0. Для установки любое ненулевое значение (обычно 1).

Отключиться от базы данных позволяет метод `disconnect()`:

```
$db->disconnect(); # Закрываем соединение
```

Для того чтобы подключиться к серверу MySQL, можно воспользоваться следующим кодом:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
```

```
my $passw = '';
my $db = DBI->connect($ds, $user, $passw)
or die("Не удалось установить подключение к базе данных. $DBI::errstr");
# Выполняем работу с базой данных
$db->disconnect(); # Закрываем соединение
```

Или альтернативным вариантом:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw, {PrintError=>0,
RaiseError=>1});
# Выполняем работу с базой данных
$db->disconnect(); # Закрываем соединение
```

Возможен и такой:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
# Выполняем работу с базой данных
$db->disconnect(); # Закрываем соединение
```

3.8.2. Выполнение запроса к базе данных

Выполнить запрос к базе данных позволяют методы `do()` и `prepare()`. Метод `do()` используется для выполнения запросов, не требующих получения данных. Например, запросов `INSERT`, `DELETE` и `UPDATE`. Метод `do()` имеет следующий формат:

```
<Идентификатор>->do(<SQL-запрос>);
```

Для примера добавим еще один город в таблицу `city`:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("INSERT INTO city VALUES(NULL, 'Новгород')");
$db->disconnect(); # Закрываем соединение
```

Метод `prepare()` используется для подготовки запроса. Исполнение подготовленного запроса осуществляется методом `execute()`. Метод `prepare()` имеет следующий формат:

```
<ID результата> = <Идентификатор>->prepare(<SQL-запрос>);
<ID результата>->execute();
```

Метод `prepare()` возвращает идентификатор результата. В случае ошибки описание ошибки доступно через свойство `errstr`, а номер ошибки через `err`. Для удаления идентификатора результата применяется метод `finish()`:

```
<ID результата>->finish();
```

Для примера получим все записи из таблицы `city`:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city");
```

```
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   # Обрабатываем результат
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Для того чтобы записи возвращались в нужной кодировке, следует после выбора базы данных указать следующий запрос:

```
$db->do("SET NAMES cp1251");
```

Метод `quote()` экранирует все специальные символы в строке и заключает строку в апострофы. Результат метода зависит от используемого драйвера базы данных.

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw) or die("Ошибка");
my $str = "Д'Артаньян и три мушкетера";
print $db->quote($str);
$db->disconnect(); # Закрываем соединение
```

Вывод:

```
'Д\'Артаньян и три мушкетера'
```

3.8.3. Обработка результата запроса

Метод `dump_results()` возвращает все записи в упорядоченном виде. Имеет следующий формат:

```
<ID результата>->dump_results([<Максимальный размер>, <Разделитель
строк>, <Разделитель полей>, <Ссылка на дескриптор файла>]);
```

Все параметры не являются обязательными:

- `<Максимальный размер>` — задает максимальный размер поля (по умолчанию 35);

- <Разделитель строк> — строка для разделения строк данных (по умолчанию \n);
- <Разделитель полей> — строка для разделения полей (по умолчанию используется запятая);
- <Ссылка на дескриптор файла> — ссылка на дескриптор файла. По умолчанию данные выводятся на STDOUT.

Например:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   $res->dump_results(50, "<BR>\n", " — ");
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
'1' — 'Санкт-Петербург'<BR>
'2' — 'Москва'<BR>
'3' — 'Новгород'
3 rows
```

Метод rows() возвращает количество полей в результате. Точные данные возвращаются только для DBD::mSQL и DBD::mysql:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   print $res->rows(); # Выведет 3
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Метод `fetchrow_array()` на каждой итерации возвращает одну строку в массив. После завершения выборки возвращается значение `undef`:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
```

```
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city ORDER BY City");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   while (my @row = $res->fetchrow_array()) {
      print $row[0], " — ", $row[1], "<BR>\n";
   }
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
2 — Москва<BR>
3 — Новгород<BR>
1 — Санкт-Петербург<BR>
```

Метод `fetchrow_arrayref()` на каждой итерации возвращает ссылку на массив:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city ORDER BY City DESC");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
```

```
else {
   while (my $row = $res->fetchrow_arrayref()) {
      print $row->[0], " — ", $row->[1], "<BR>\n";
   }
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
1 — Санкт-Петербург<BR>
3 — Новгород<BR>
2 — Москва<BR>
```

Метод `fetch()` на каждой итерации возвращает ссылку на массив:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city WHERE id_City > 1");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   while (my $row = $res->fetch()) {
      print $row->[0], " — ", $row->[1], "<BR>\n";
   }
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
2 — Москва<BR>
3 — Новгород<BR>
```

Метод `fetchrow_hashref()` на каждой итерации возвращает ссылку на ассоциативный массив:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   while (my $row = $res->fetchrow_hashref()) {
      print $row->{'id_City'}, " — ", $row->{'City'}, "<BR>\n";
   }
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
1 — Санкт-Петербург<BR>
2 — Москва<BR>
3 — Новгород<BR>
```

Метод `fetchall_arrayref()` возвращает массив, содержащий ссылки на массивы с результатами запроса:

```
use DBI;
my $ds = 'DBI:mysql:tests:localhost';
my $user = 'root';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
$db->do("SET NAMES cp1251");
my $res = $db->prepare("SELECT * FROM city");
$res->execute();
if ($res->err) {
   # Если возникла ошибка
   print "Ошибка " . $res->errstr;
}
else {
   my $rows = $res->fetchall_arrayref();
   print "Всего записей: ", scalar(@$rows), "<BR>\n";
   foreach my $row (@$rows) {
      print $row->[0], " — ", $row->[1], "<BR>\n";
   }
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод в исходном HTML-коде:

```
Всего записей: 3<BR>
1 — Санкт-Петербург<BR>
2 — Москва<BR>
3 — Новгород<BR>
```

3.8.4. Использование заполнителей

Как уже говорилось, для запросов `INSERT`, `DELETE` и `UPDATE` обычно применяют метод `do()`, ведь вполне достаточно знать лишь количество обновленных полей таблицы. Тем не менее, это справедливо только в случае одиночных за-

просов. Если необходимо произвести обновление несколько раз (например, добавить несколько городов в таблицу `city`), то использование метода `do()` приводит к снижению производительности скрипта. Для примера рассмотрим добавление четырех городов в таблицу `city`:

```
$db->do("INSERT INTO city VALUES (NULL, 'Саратов')");
$db->do("INSERT INTO city VALUES (NULL, 'Омск')");
$db->do("INSERT INTO city VALUES (NULL, 'Тверь')");
$db->do("INSERT INTO city VALUES (NULL, 'Иркутск')");
```

Этот фрагмент кода можно записать так:

```
my $res1 = $db->prepare("INSERT INTO city VALUES (NULL, 'Саратов')");
$res1->execute();
my $res2 = $db->prepare("INSERT INTO city VALUES (NULL, 'Омск')");
$res2->execute();
my $res3 = $db->prepare("INSERT INTO city VALUES (NULL, 'Тверь')");
$res3->execute();
my $res4 = $db->prepare("INSERT INTO city VALUES (NULL, 'Иркутск')");
$res4->execute();
```

Все дело в том, что метод `do()` — это последовательный вызов методов `prepare()` и `execute()`. Метод `prepare()` подготавливает запрос, а метод `execute()` его выполняет. Иными словами, чтобы вставить четыре записи, мы четыре раза подготавливали запрос, а затем его выполняли. С помощью заполнителей мы можем один раз подготовить запрос, а затем четыре раза его выполнить, передавая новые города в методе `execute()`. Запрос подготавливается и выполняется следующим образом:

```
my $result = $db->prepare("INSERT INTO city VALUES (NULL, ?)");
$result->execute("Саратов");
$result->execute("Омск");
$result->execute("Тверь");
$result->execute("Иркутск");
```

В этом примере символ `?` является заполнителем. В момент выполнения запроса вместо символа `?` подставляется название города.

Если в запросе несколько символов `?`, то в методе `execute()` значения указываются через запятую:

```
my $result = $db->prepare("INSERT INTO Tab VALUES (NULL, ?, ?, ?)");
$result->execute($param1, $param2, $param3);
```

3.8.5. Особенности доступа к базе данных Access

Как вы уже знаете, модуль DBI предоставляет единый интерфейс доступа ко всем базам данных. Поэтому все изученные методы применимы и для доступа к базе данных Access. В этом разделе мы рассмотрим некоторые особенности работы с базой данных Access.

Сначала необходимо создать новую базу данных. Для этого запускаем программу Microsoft Access. В открывшемся окне устанавливаем флажок **Новая база данных**. Нажимаем кнопку **OK**. Выбираем папку C:\WebServers\home \perlbook.ru. В поле **Имя файла** вводим testDBAccess. Нажимаем кнопку **Создать**.

Теперь создадим таблицу. Для этого выбираем пункт **Создание таблицы в режиме конструктора**. В поле **Имя поля** вводим id, а из списка **Тип данных** выбираем пункт **Счетчик**. Щелкаем правой кнопкой мыши на созданной строке и из контекстного меню выбираем пункт **Ключевое поле**. Переходим на вторую строку. В поле **Имя поля** вводим text, а из списка **Тип данных** выбираем пункт **Поле MEMO**. Сохраняем таблицу под именем Tab1. Открываем созданную таблицу и в поле **text** вводим следующий текст:

```
Модуль DBI предоставляет единый интерфейс доступа ко всем базам данных.
Поэтому все методы применимы и для доступа к базе данных Access.
```

Закрываем программу Microsoft Access.

Теперь необходимо зарегистрировать созданную базу данных в качестве источника данных ODBC. Для этого выбираем пункт меню **Пуск | Настройка | Панель управления** (или **Пуск | Панель управления**). В открывшемся окне выбираем пункт **Администрирование**. Далее выбираем пункт **Источники данных (ODBC)**. Переходим на вкладку **Системный DSN**. Нажимаем кнопку **Добавить**. В открывшемся окне выбираем пункт **Microsoft Access Driver (*.mdb)** и нажимаем кнопку **Готово**. В открывшемся окне **Установка драйвера ODBC для Microsoft Access** нажимаем кнопку **Выбрать**. Находим нашу базу данных testDBAccess.mdb и нажимаем кнопку **OK**. В поле **Имя источника данных** вводим testDBAccess. Нажимаем **OK**. Созданный источник данных ODBC отобразится на вкладке **Системный DSN**. Закрываем окно **Администратор источников данных ODBC**.

Теперь попробуем получить данные из созданного источника данных ODBC:

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use DBI;
print "Content-type: text/html\n\n";
```

```
my $ds = 'DBI:ODBC:testDBAccess';
my $user = '';
my $passw = '';
my $db = DBI->connect($ds, $user, $passw);
if (!$db) {
   print "Не удалось установить подключение к базе данных.";
   print "<BR>$DBI::err"; # Номер ошибки
   print "<BR>$DBI::errstr"; # Текст ошибки
   exit();
}
my $res = $db->prepare("SELECT * FROM Tab1");
$res->execute() or die("Проблема " . $db->errstr());
while (my @row = $res->fetchrow_array()) {
   print $row[0], " — ", $row[1], "<BR>\n";
}
if ($db->err()) { # Если возникла ошибка
   print  $db->errstr();
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

В результате выполнения программы получим сообщение об ошибке:

```
[Microsoft][Драйвер ODBC Microsoft Access]Усечение данных строки справа
на строке номер 2 (text) (SQL-01004)
```

Именно в этом и состоит основная особенность работы с базой данных Access. Все дело в том, что при выборке данных из больших полей типа `MEMO` существует ограничение на количество возвращаемых символов — длина не превышает 80 символов. Если данные имеют большую длину, то по умолчанию генерируется ошибка усечения поля. За поведение программы при усечении поля отвечает атрибут `LongTruncOk`. Если атрибуту присвоить значение 1, то ошибка выводиться не будет, а данные будут обрезаны:

```
# ...
$db->{LongTruncOk} = 1;
my $res = $db->prepare("SELECT * FROM Tab1");
$res->execute() or die("Проблема " . $db->errstr());
while (my @row = $res->fetchrow_array()) {
   print $row[0], " — ", $row[1], "<BR>\n";
}
# ...
```

Вывод:

```
1 — Модуль DBI предоставляет единый интерфейс доступа ко всем базам
данных. Поэтому
```

Как видно из примера, мы не получили значение поля полностью. Чтобы получить все данные, необходимо указать максимальный размер поля в атрибуте `LongReadLen`:

```
# ...
$db->{LongReadLen} = 200;
$db->{LongTruncOk} = 1;
my $res = $db->prepare("SELECT * FROM Tab1");
$res->execute() or die("Проблема " . $db->errstr());
while (my @row = $res->fetchrow_array()) {
   print $row[0], " — ", $row[1], "<BR>\n";
}
# ...
```

Вывод:

```
1 — Модуль DBI предоставляет единый интерфейс доступа ко всем базам
данных. Поэтому все методы применимы и для доступа к базе данных Access.
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Не следует указывать слишком большое значение атрибута `LongReadLen`, он оказывает непосредственное влияние на объем памяти, используемом программой.

3.9. Аутентификация с помощью Perl. Создание Личного кабинета

Модуль `CGI::Session` позволяет произвести аутентификацию посетителей при помощи механизма сессий. Для начала необходимо установить модуль. Подключаемся к Интернету и запускаем Денвер. Переходим на диск Z и запускаем файл Z:\usr\local\perl\bin\ppm-shell.bat. В командной строке должно быть приглашение:

```
ppm>
```

Набираем команду:

```
install CGI-Session
```

и нажимаем клавишу <Enter>.

ОБРАТИТЕ ВНИМАНИЕ

Файл ppm-shell.bat должен быть обязательно запущен с диска Z. В противном случае модуль будет установлен в каталог C:\usr, а не в C:\WebServers\usr.

3.9.1. Сохранение данных сессии в файле

Модуль `CGI::Session` предоставляет следующие методы:

- `new()` — создает новую сессию. Метод имеет следующий формат:

  ```
  new(<Метод>, <Идентификатор>, <Параметры>)
  ```

  Параметр `<Метод>` определяет место хранения данных сеанса. Данные могут сохраняться как в файле, так и в базе данных. Для использования файлов следует указать следующую строку:

  ```
  driver:file
  ```

  Для создания новой сессии в параметре `<Идентификатор>` следует указать значение `undef`. Если в качестве значения передать существующий идентификатор сессии, то будет использована уже существующая сессия, а не создана новая.

  При использовании файлов в параметре `<Параметры>` можно указать местоположение файлов, в которых будут сохраняться данные сессий:

  ```
  my $sess = CGI::Session->new("driver:file", undef,
  {Directory=>'tmp'});
  ```

- `errstr()` — возвращает сообщение об ошибке:

  ```
  my $sess = CGI::Session->new("driver:file", undef, {Directory=>'tmp'})
  or die CGI::Session->errstr();
  ```

- `id()` — возвращает идентификатор сессии:

  ```
  my $sess = CGI::Session->new("driver:file", undef, {Directory=>'tmp'});
  my $SID = $sess->id();
  ```

- `name()` — возвращает или задает имя идентификатора сессии. По умолчанию используется значение `CGISESSID`:

  ```
  $sess->name('SID');
  ```

- `expire()` — задает время жизни сессии. Время отсчитывается от значения метода `atime()`. В качестве параметра указывается относительное значение:

  - `+30s` — 30 секунд;
  - `+15m` — 15 минут;
  - `+3h` — 3 часа;

- `+2d` — 2 дня;
- `+1w` — 1 неделя;
- `+1M` — 1 месяц;
- `+1y` — 1 год.

Например:

```
$sess->expire("+5m");
```

- `etime()` — возвращает время жизни сессии в секундах:

```
print $sess->etime();
```

- `is_expired()` — возвращает `true`, если истекло время жизни сессии:

```
if ($sess->is_expired()) {
   # Время сессии истекло
}
```

- `ctime()` — задает или возвращает время создания сессии в секундах;
- `atime()` — задает или возвращает время последнего доступа к сессии в секундах. От значения метода `atime()` отсчитывается время жизни сессии:

```
$sess->atime(time());
```

- `is_empty()` — возвращает `true`, если сессия пустая;
- `remote_addr()` — возвращает IP-адрес, сохраненный в сессии. С помощью этого метода мы можем проверить соответствие IP-адресов:

```
if ($sess->remote_addr() ne $ENV{'REMOTE_ADDR'}) {
# IP-адреса не совпадают
}
```

- `param()` — позволяет сохранить значения переменных в сессии, а также получить эти значения в дальнейшем. Для получения значений используется следующий формат:

```
param(<Имя переменной>)
param(-name=><Имя переменной>)
```

Например:

```
my $formLogin = $sess->param('Login');
```

Для сохранения данных используется следующий формат:

```
param(<Имя переменной>, <Значение переменной>)
param(-name=><Имя переменной>, -value=><Значение переменной>)
```

Например:

```
$sess->param(-name=>'Login', -value=>'nik');
```

- `clear()` — позволяет удалить данные из сессии;
- `flush()` — сохраняет данные сессии из буфера в файл. Вызывается автоматически при удалении объекта, а также при вызове метода `close()`:

```
$sess->flush();
```

- `delete()` — удаляет сессию:

```
$sess->delete();
$sess->flush();
```

При запуске сессии на компьютер пользователя необходимо установить cookies с именем сессии, заданным методом `name()`, и значением вида `db711b560810e7f90d67a4c8e6a873af`. В папке для временных файлов сессий будет создан файл с именем `cgisess_db711b560810e7f90d67a4c8e6a873af`.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Переменная, зарегистрированная в рамках сессии, сохраняется не в cookies пользователя, а в специальном файле на сервере. В cookies пользователя сохраняется только переменная с именем сессии и значением вида `db711b560810e7f90d67a4c8e6a873af`.

Если прием cookies отключен в настройках Web-браузера, то к ссылкам необходимо добавить идентификатор сессии. Например:

```
?SID=db711b560810e7f90d67a4c8e6a873af
```

Если на странице имеется форма, то при отключенных cookies внутрь формы необходимо добавить скрытое поле. Например:

```
<input type="hidden" name="SID" value="db711b560810e7f90d67a4c8e6a873af">
```

При первом запуске мы устанавливаем cookies, а также указываем имя сессии в URL-адресе для всех внутренних ссылок. Если cookies разрешено использовать, то в дальнейшем добавление к URL необходимо прекратить.

Для примера создадим папку user в папке C:\WebServers\home\perlbook.ru \cgi-bin. В этой папке создадим папку tmp (для хранения файлов сессий) и следующие файлы:

- index.pl — содержит форму для ввода логина и пароля (листинг 3.57);
- secure.pl — файл с информацией только для прошедших аутентификацию пользователей (листинг 3.58);
- exit.pl — для завершения сеанса (листинг 3.59);
- MyData.pl — для хранения логина и пароля (листинг 3.60).

Листинг 3.57. Содержимое файла index.pl

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
use Digest::MD5 qw( md5_hex );
require "MyData.pl";
my $log = $MyData::enter_login;
my $pass = $MyData::enter_passw;
my $err = "";
my $formLogin = param('login');
my $formPass = param('passw');

if (defined($formLogin) && defined($formPass)) {
   $formPass = md5_hex($formPass);
   if ($formLogin eq $log && $formPass eq $pass) {
      my $tmp = "C:/WebServers/home/perlbook.ru/cgi-bin/user/tmp/";
    my $sess = CGI::Session->new("driver:file", undef, {Directory=>$tmp})
                        or die CGI::Session->errstr();
      $sess->name('SID');
      my $cookies = cookie(-name=>$sess->name(), -value=>$sess->id(),
                           -path=>'/', -domain=>'.perlbook.ru');
      print "Set-Cookie: $cookies\n";
      $sess->param(-name=>'Login', -value=>$formLogin);
      $sess->param(-name=>'Password', -value=>$formPass);
      $sess->expire("+5m");
      $sess->flush();
      print "Location: secure.pl?SID=" . $sess->id() . "\n\n";
      exit();
   }
   else {
      $err = "<CENTER><FONT color=\"red\"><B>\n";
      $err .= "Логин или пароль введены неправильно!\n";
      $err .= "</B></FONT></CENTER>\n";
   }
}
```

```
print "Content-type: text/html\n\n";
print <<HTML_COD;
<HTML>
<HEAD>
<TITLE>Вход с систему</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<TABLE border="0" cellspacing="0" width="100%" height="100%">
<TR><TD align="center" valign="middle">
<FORM method="POST" action="http://perlbook.ru/cgi-bin/user/index.pl">
<TABLE border="0" cellspacing="0" width="200">
<CAPTION><B>Вход в систему</B>
</CAPTION>
<TR><TD align="right"><B>Логин:</B></TD>
<TD><INPUT type="text" name="login"></TD></TR>
<TR><TD align="right"><B>Пароль:</B></TD>
<TD><INPUT type="password" name="passw"></TD></TR>
<TR>
<TD align="center" colspan="2">
<INPUT type="submit" value="Войти">
</TD></TR></TABLE>
</FORM>
$err
</TD></TR></TABLE>
</BODY>
</HTML>
HTML_COD
# Конец скрипта
```

Листинг 3.58. Содержимое файла secure.pl

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
my $SID;
my $method;
```

```
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
   $method = 'cookies';
}
elsif (defined(param('SID'))) {
   $SID = param('SID');
   $method = 'path';
}
else {
   print "Location: index.pl\n\n";
   exit();
}
my $tmp_path = "C:/WebServers/home/perlbook.ru/cgi-bin/user/tmp/";
my $sess = CGI::Session->new("driver:file", $SID, {Directory=>$tmp_path})
                  or die CGI::Session->errstr();
$sess->name('SID');
if ($sess->is_empty()) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
# Если время сессии истекло
if ($sess->is_expired()) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
# Если IP-адреса не совпадают
if ($sess->remote_addr() ne $ENV{'REMOTE_ADDR'}) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
require "MyData.pl";
my $log = $MyData::enter_login;
my $pass = $MyData::enter_passw;
my $formLogin = $sess->param('Login');
my $formPass = $sess->param('Password');
```

```
if (defined($formLogin) && defined($formPass)) {
   unless ($formLogin eq $log && $formPass eq $pass) {
      &f_delete_session();
      print "Location: index.pl\n\n";
      exit();
   }
}
else {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}

print "Content-type: text/html\n\n";
print "<HTML>\n";
print "<HEAD>\n";
print "<TITLE>Информация для прошедших аутентификацию</TITLE>\n";
print "</HEAD>\n";
print "<BODY bgcolor=\"#FFFFFF\">\n";
print "Информация для прошедших аутентификацию<BR><BR>\n";
print "<A href=\""  . f_add_sid("exit.pl");
print "\">Выйти из системы</A><BR>\n";
print "<FORM action=\"http://perlbook.ru/cgi-bin/user/exit.pl\">\n";
if ($method eq 'path') {
   print "<INPUT type=\"hidden\" name=\"" . $sess->name() . "\" ";
   print "value=\"$SID\">\n";
}
print "<INPUT type=\"submit\" value=\"Выйти\">\n";
print "</FORM>\n";

print "</BODY>\n";
print "</HTML>\n";
# Продлеваем время сессии
$sess->atime(time());
$sess->flush();
# Удаление сессии
sub f_delete_session {
   my $cookies = cookie(-name=>$sess->name(), -value=>'',
      -expires=>'-1d', -path=>'/', -domain=>'.perlbook.ru');
```

```
   print "Set-Cookie: $cookies\n";
   $sess->delete();
   $sess->flush();
}
# Добавление SID к URL-адресу
sub f_add_sid {
   my $url = shift();
   if ($method eq 'path') {
      if (defined($SID) && $url !~ m/SID=/) {
         if ($url !~ m/\?/) {
            $url .= "?SID=" . $SID;
         }
         else {
            $url .= "&SID=" . $SID;
         }
      }
   }
   return $url;
}
```

Листинг 3.59. Содержимое файла exit.pl

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
my $SID;
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
}
elsif (defined(param('SID'))) {
   $SID = param('SID');
}
else {
   print "Location: index.pl\n\n";
   exit();
}
```

```
my $tmp_path = "C:/WebServers/home/perlbook.ru/cgi-bin/user/tmp/";
my $sess = CGI::Session->new("driver:file", $SID, {Directory=>$tmp_path})
                  or die CGI::Session->errstr();
$sess->name('SID');
my $cookies = cookie(-name=>$sess->name(), -value=>'', -expires=>'-1d',
                        -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies\n";
$sess->delete();
$sess->flush();
print "Location: index.pl\n\n";
exit();
```

Листинг 3.60. Содержимое файла MyData.pl

```
package MyData;

our $enter_login = "login"; # Логин
# Пароль
our $enter_passw = "202cb962ac59075b964b07152d234b70";
1;
```

В данном примере используется только один логин (`login`) и пароль (`123`). В реальной практике для каждого пользователя создается свой логин и пароль. Для хранения учетных записей используется файл или чаще всего база данных. Если используется обычный файл (как в нашем случае), то он должен содержать пароль в зашифрованном виде, а сам файл должен быть недоступен через Интернет. В нашем примере файл MyData.pl должен быть расположен вне папки Web-документов, а не C:\WebServers\home\perlbook.ru \cgi-bin\user.

3.9.2. Удаление просроченных файлов

Далеко не все пользователи переходят по ссылке **Выйти из системы**. По этой причине файлы с данными сессий накапливаются в папке. Время от времени эти просроченные файлы необходимо удалять. Для этой цели можно использовать метод `find()` из модуля `CGI::Session`. Метод имеет следующий формат:

```
find(<Метод>, <Ссылка на функцию>, <Параметры>)
```

Назначения параметров `<Метод>` и `<Параметры>` идентичны параметрам метода `new()`. Во втором параметре указывается ссылка на подпрограмму обработки.

На каждой итерации этой подпрограмме передается ссылка на экземпляр класса `CGI::Session` (листинг 3.61).

Листинг 3.61. Удаление просроченных файлов

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use CGI::Session;
print "Content-type: text/html\n\n";
my $tmp_path = "C:/WebServers/home/perlbook.ru/cgi-bin/user/tmp/";
CGI::Session->find("driver:file", \&f_delete, { Directory=>$tmp_path } );
print "Операция произведена<BR>";
sub f_delete {
   my ($session) = @_;
   next if $session->is_empty();
   if ( ($session->ctime() + 86400) < time() ) {
      $session->delete();
   }
}
```

Данную программу следует время от времени запускать либо вручную, либо по расписанию (сервис `cron` в UNIX).

3.9.3. Сохранение данных сессии в базе данных MySQL

В предыдущем разделе мы рассмотрели сохранение данных сессии в файле. Помимо файлов модуль `CGI::Session` позволяет использовать базу данных MySQL. Для создания новой сессии используется метод `new()` следующего формата:

```
new("driver:mysql", <Идентификатор>, <Параметры>)
```

Для создания новой сессии в параметре `<Идентификатор>` следует указать значение `undef`. Если в качестве значения передать существующий идентификатор сессии, то будет использована уже существующая сессия, а не создана новая.

В блоке `<Параметры>` можно указать:

- `DataSource` — строка для подключения к MySQL. Указывается следующим образом:

  ```
  DataSource => "dbi:mysql:<Название базы данных>:<Хост>"
  ```

Например:

```
DataSource => "dbi:mysql:SessionDB:localhost"
```

- `User` — имя пользователя;
- `Password` — пароль.

Например:

```
my $sess = CGI::Session->new("driver:mysql", undef,
           { DataSource => "dbi:mysql:SessionDB:localhost",
             User => "Cemen", Password => "123456"})
             or die CGI::Session->errstr();
```

Если подключение к базе данных MySQL было установлено ранее, то можно просто указать дескриптор соединения:

```
my $sess = CGI::Session->new("driver:mysql", undef,
           { Handle => $db })
             or die CGI::Session->errstr();
```

По умолчанию модуль `CGI::Session` использует таблицу с названием `sessions`. Для изменения названия таблицы следует указать это название в переменной `TABLE_NAME` модуля `CGI::Session::MySQL`:

```
$CGI::Session::MySQL::TABLE_NAME = 'MySessions';
```

Прежде чем рассматривать пример, необходимо создать базу данных, пользователя базы данных и, наконец, таблицу для хранения данных сессии. Для этой цели можно воспользоваться программой phpMyAdmin. Запускаем Денвер и открываем Web-браузер. В адресной строке набираем следующее:

```
http://localhost/Tools/phpMyAdmin/
```

В поле **Создать новую БД** вводим `SessionDB`. Из списка **Сравнение** выбираем пункт **cp1251_general_ci**. Нажимаем кнопку **Создать**. Отобразится сообщение: **БД SessionDB была создана**.

Для создания таблицы переходим на вкладку **SQL**. В поле **Выполнить SQL запрос(ы) на БД** вводим SQL-команду:

```
CREATE TABLE MySessions (
   id CHAR(32) NOT NULL PRIMARY KEY,
   a_session TEXT NOT NULL
) TYPE = MYISAM CHARACTER SET cp1251 COLLATE cp1251_general_ci;
```

Нажимаем кнопку **Пошел**. Название новой таблицы отобразится в левой части окна Web-браузера в виде ссылки. Если перейти по этой ссылке, то в правой части отобразится структура таблицы.

Теперь создадим пользователя базы данных. Для этого переходим по ссылке **localhost** в верхней части окна. Далее переходим по ссылке **Привилегии**. Затем переходим по ссылке **Добавить нового пользователя**. В поле **Имя пользователя** вводим `Cemen`. Из списка **Хост** выбираем пункт **Local**. В поле **Пароль** вводим значение `123456`, а затем повторяем его в поле **Подтверждение**. Ставим все флажки в разделах **Данные** и **Структура**. Нажимаем кнопку **Пошел**. Отобразится сообщение: **Был добавлен новый пользователь**. Из списка **Добавить привилегии на следующую базу** выбираем пункт **SessionDB**. Ставим все флажки в разделах **Данные** и **Структура**. Нажимаем кнопку **Пошел**. Далее необходимо перезагрузить привилегии. Для этого переходим по ссылке **localhost** в верхней части окна Web-браузера. Затем по ссылке **Привилегии**. В открывшемся окне переходим по ссылке **Перезагрузить привилегии**. Отобразится сообщение: **Привилегии были успешно перезагружены**.

Как и в случае с файлами, в папке user создадим следующие файлы:

- index.pl — содержит форму для ввода логина и пароля (листинг 3.62);
- secure.pl — файл с информацией только для прошедших аутентификацию пользователей (листинг 3.63);
- exit.pl — для завершения сеанса (листинг 3.64);
- MyData.pl — для хранения логина и пароля, а таже данных для подключения к MySQL и времени жизни сессии (листинг 3.65).

Листинг 3.62. Содержимое файла index.pl

```
#!/usr/bin/perl -w
use strict;
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
use Digest::MD5 qw( md5_hex );
require "MyData.pl";
my $log = $MyData::enter_login;
my $pass = $MyData::enter_passw;
my $err = "";
my $formLogin = param('login');
my $formPass = param('passw');

if (defined($formLogin) && defined($formPass)) {
   $formPass = md5_hex($formPass);
```

```
   if ($formLogin eq $log && $formPass eq $pass) {
      $CGI::Session::MySQL::TABLE_NAME = $MyData::TableName;
      my $sess = CGI::Session->new("driver:mysql", undef,
                 { DataSource => $MyData::DataSource,
                 User => $MyData::User, Password => $MyData::Password })
                 or die CGI::Session->errstr();
      $sess->name('SID');
      my $cookies = cookie(-name=>$sess->name(), -value=>$sess->id(),
                           -path=>'/', -domain=>'.perlbook.ru');
      print "Set-Cookie: $cookies\n";
      $sess->param(-name=>'Login', -value=>$formLogin);
      $sess->param(-name=>'Password', -value=>$formPass);
      $sess->expire($MyData::SessTime);
      $sess->flush();
      print "Location: secure.pl?SID=" . $sess->id() . "\n\n";
      exit();
   }
   else {
      $err = "<CENTER><FONT color=\"red\"><B>\n";
      $err .= "Логин или пароль введены неправильно!\n";
      $err .= "</B></FONT></CENTER>\n";
   }
}
print "Content-type: text/html\n\n";
print <<HTML_COD;
<HTML>
<HEAD>
<TITLE>Вход с систему</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF">
<TABLE border="0" cellspacing="0" width="100%" height="100%">
<TR><TD align="center" valign="middle">
<FORM method="POST" action="http://perlbook.ru/cgi-bin/user/index.pl">
<TABLE border="0" cellspacing="0" width="200">
<CAPTION><B>Вход в систему</B>
</CAPTION>
<TR><TD align="right"><B>Логин:</B></TD>
```

```
<TD><INPUT type="text" name="login"></TD></TR>
<TR><TD align="right"><B>Пароль:</B></TD>
<TD><INPUT type="password" name="passw"></TD></TR>
<TR>
<TD align="center" colspan="2">
<INPUT type="submit" value="Войти">
</TD></TR></TABLE>
</FORM>
$err
</TD></TR></TABLE>
</BODY>
</HTML>
HTML_COD
# Конец скрипта
```

Листинг 3.63. Содержимое файла secure.pl

```
#!/usr/bin/perl -w
use strict;
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
require "MyData.pl";
my $SID;
my $method;
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
   $method = 'cookies';
}
elsif (defined(param('SID'))) {
   $SID = param('SID');
   $method = 'path';
}
else {
   print "Location: index.pl\n\n";
   exit();
}
```

```
$CGI::Session::MySQL::TABLE_NAME = $MyData::TableName;
my $sess = CGI::Session->new("driver:mysql", $SID,
                  { DataSource => $MyData::DataSource,
                  User => $MyData::User, Password => $MyData::Password })
                  or die CGI::Session->errstr();
$sess->name('SID');
if ($sess->is_empty()) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
# Если время сессии истекло
if ($sess->is_expired()) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
# Если IP-адреса не совпадают
if ($sess->remote_addr() ne $ENV{'REMOTE_ADDR'}) {
   &f_delete_session();
   print "Location: index.pl\n\n";
   exit();
}
require "MyData.pl";
my $log = $MyData::enter_login;
my $pass = $MyData::enter_passw;
my $formLogin = $sess->param('Login');
my $formPass = $sess->param('Password');
if (defined($formLogin) && defined($formPass)) {
   unless ($formLogin eq $log && $formPass eq $pass) {
      &f_delete_session();
      print "Location: index.pl\n\n";
      exit();
   }
}
else {
   &f_delete_session();
```

```
   print "Location: index.pl\n\n";
   exit();
}

print "Content-type: text/html\n\n";
print "<HTML>\n";
print "<HEAD>\n";
print "<TITLE>Информация для прошедших аутентификацию</TITLE>\n";
print "</HEAD>\n";
print "<BODY bgcolor=\"#FFFFFF\">\n";
print "Информация для прошедших аутентификацию<BR><BR>\n";
print "<A href=\""  . f_add_sid("exit.pl");
print "\">Выйти из системы</A><BR>\n";
print "<FORM action=\"http://perlbook.ru/cgi-bin/user/exit.pl\">\n";
if ($method eq 'path') {
   print "<INPUT type=\"hidden\" name=\"" . $sess->name() . "\" ";
   print "value=\"$SID\">\n";
}
print "<INPUT type=\"submit\" value=\"Выйти\">\n";
print "</FORM>\n";

print "</BODY>\n";
print "</HTML>\n";
# Продлеваем время сессии
$sess->atime(time());
$sess->flush();
# Удаление сессии
sub f_delete_session {
   my $cookies = cookie(-name=>$sess->name(), -value=>'',
      -expires=>'-1d', -path=>'/', -domain=>'.perlbook.ru');
   print "Set-Cookie: $cookies\n";
   $sess->delete();
   $sess->flush();
}
# Добавление SID к URL-адресу
sub f_add_sid {
   my $url = shift();
```

```
   if ($method eq 'path') {
      if (defined($SID) && $url !~ m/SID=/) {
         if ($url !~ m/\?/) {
            $url .= "?SID=" . $SID;
         }
         else {
            $url .= "&SID=" . $SID;
         }
      }
   }
   return $url;
}
```

Листинг 3.64. Содержимое файла exit.pl

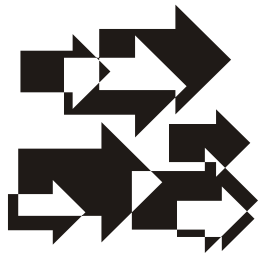
```
#!/usr/bin/perl -w
use strict;
use CGI::Carp qw(fatalsToBrowser);
use CGI qw( :standard);
use CGI::Session;
require "MyData.pl";
my $SID;
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
}
elsif (defined(param('SID'))) {
   $SID = param('SID');
}
else {
   print "Location: index.pl\n\n";
   exit();
}
$CGI::Session::MySQL::TABLE_NAME = $MyData::TableName;
my $sess = CGI::Session->new("driver:mysql", $SID,
                  { DataSource => $MyData::DataSource,
                  User => $MyData::User, Password => $MyData::Password })
                  or die CGI::Session->errstr();
```

```
$sess->name('SID');
my $cookies = cookie(-name=>$sess->name(), -value=>'', -expires=>'-1d',
                        -path=>'/', -domain=>'.perlbook.ru');
print "Set-Cookie: $cookies\n";
$sess->delete();
$sess->flush();
print "Location: index.pl\n\n";
exit();
```

Листинг 3.65. Содержимое файла MyData.pl

```
package MyData;
# Логин
our $enter_login = "login";
# Пароль
our $enter_passw = "202cb962ac59075b964b07152d234b70";
# Данные для подключения к MySQL
our $TableName = 'MySessions';
our $DataSource = "dbi:mysql:SessionDB:localhost";
our $User = "Cemen";
our $Password = "123456";
# Время жизни сессии
our $SessTime = "+15m";
1;
```

ГЛАВА 4



Основы MySQL

4.1. Создание базы данных

MySQL — это система управления реляционными базами данных. Сервер MySQL позволяет эффективно работать с данными и обеспечивает быстрый доступ к данным одновременно нескольким пользователям. При этом доступ к данным предоставляется только пользователям, имеющим на это право.

Что же такое база данных? *Реляционная база данных* — это совокупность двумерных таблиц, связанных отношениями друг с другом. Каждая *таблица* содержит совокупность записей. В свою очередь, *запись* — это набор полей, содержащих связанную информацию. Любое *поле* в базе данных имеет имя и определенный тип. Имя таблицы должно быть уникальным в пределах базы данных. В свою очередь, имя поля должно быть уникальным в пределах таблицы.

Прежде чем рассматривать создание реляционных баз данных, мы установим на компьютер пакет расширения MySQL5. Для этого останавливаем Денвер (если он был запущен ранее). Со страницы **http://www.denwer.ru/packages/mysql5.html** скачиваем пакет расширения и запускаем файл Denwer3_MySQL5_2008-01-13_5.0.45.exe. Отобразится окно с запросом: **Вы действительно хотите установить пакет расширения?**. Для запуска инсталлятора нажимаем кнопку **Да**. Программа установки произведет поиск базового пакета Денвера. Если базовый пакет будет найден, то инсталлятор предложит установить пакет расширения в тот же каталог. Для подтверждения нажимаем клавишу <y>, а затем клавишу <Enter>, для обозначения начала копирования файлов и завершения установки.

Инсталлятор установит множество полезных утилит, например, mysqldump, mysqlshow и др. Так как все эти утилиты запускаются с помощью командной строки, то необходимо изучить способ их запуска. Для начала следует запус-

тить Денвер. Далее запускаем программу cmd.exe. Выбираем пункт меню **Пуск | Выполнить...** и в окне **Запуск программы** в поле **Открыть** набираем `cmd`, а затем нажимаем кнопку **Ok**. Далее необходимо сменить текущий диск на диск Z. Для этого в командной строке набираем команду:

```
Z:
```

Нажимаем клавишу <Enter>. В командной строке должно быть приглашение:

```
Z:\>
```

> ***Обратите внимание***
>
> Запускать утилиты необходимо обязательно с виртуального диска Z. Если запустить утилиту с диска C, то большинство пользователей Денвера получат сообщение об ошибке: "Character set 'cp1251' is not a compiled character secified in the '\usr\local\mysql5\share\charsets\Index.xml' file".

Теперь необходимо сделать текущим каталог Z:\usr\local\mysql5\bin\. Для этого набираем команду:

```
cd \usr\local\mysql5\bin\
```

В командной строке должно быть приглашение:

```
Z:\usr\local\mysql5\bin>
```

Теперь можно запускать утилиты.

Для примера запустим программу для управления MySQL из командной строки. В командной строке набираем команду:

```
mysql_run_to_import_dumps
```

В итоге отобразится приветствие сервера, как на рис. 4.1.

В качестве примера выведем все зарегистрированные в системе базы данных. В командной строке набираем команду:

```
SHOW DATABASES;
```

Результат выполнения команды будет представлен в виде таблицы с перечнем всех баз данных. Чтобы вывести результат в виде списка, необходимо после SQL-команды указать флаг `\G`. Например:

```
SHOW DATABASES\G
```

Для выхода из программы MySQL monitor набираем команду:

```
exit;
```

> ***Обратите внимание***
>
> Все SQL-команды должны заканчиваться точкой с запятой или флагами `\g` или `\G`. До момента ввода этих символов и нажатия клавиши <Enter> система будет в режиме ввода.

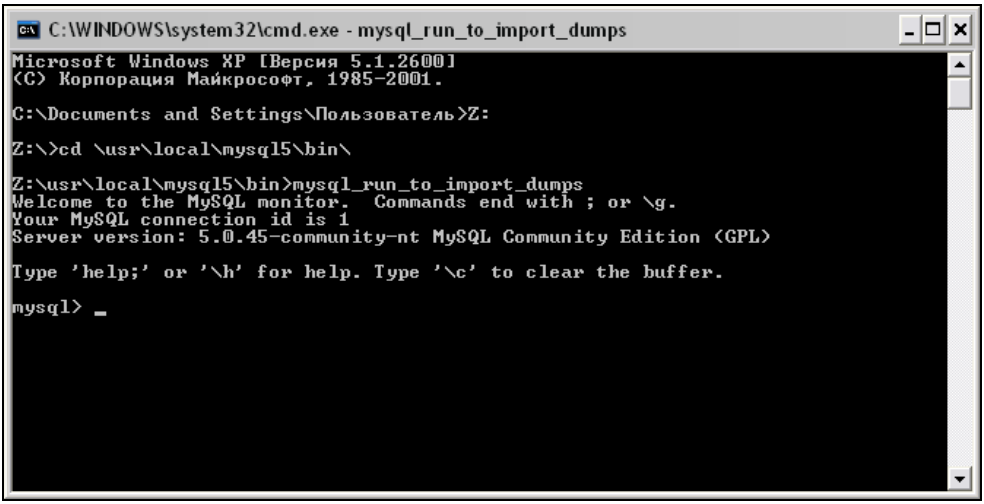


Рис. 4.1. Результат выполнения команды mysql_run_to_import_dumps

Программой MySQL monitor мы будем пользоваться очень часто. По этой причине лучше на Рабочем столе создать файл с названием MySQL_ monitor.bat. Содержимое файла должно выглядеть следующим образом:

```
@echo off
Z:
cd \usr\local\mysql5\bin\
start mysql_run_to_import_dumps -uroot
```

В последней строке с помощью параметра `-u` мы указываем пользователя, от имени которого выполняется подключение. В данном случае указывается привилегированный пользователь `root`. Если пользователю назначен пароль, то следует указать его в параметре `-p`. Например:

```
start mysql_run_to_import_dumps -uroot -p123
```

После создания файла запустить программу MySQL monitor можно с помощью двойного щелчка мыши на названии файла.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В командной строке Windows используется кодировка DOS (cp866). Чтобы символы русского алфавита отображались нормально, необходимо после запуска MySQL monitor выполнить SQL-команду `SET NAMES 'cp866';`.

Теперь можно рассматривать создание реляционных баз данных. Для начала рассмотрим таблицу заказов (табл. 4.1).

Таблица 4.1. *Таблица заказов*

Name	Address	City	Phone	Tovar	Date_orders	Price	Col	Sum
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	HDD	2007-06-20	3400	1	3400
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2007-06-20	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Монитор	2007-06-25	7200	1	7200
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Тюнер	2007-06-30	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Дискета	2007-07-01	10	10	100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Сканер	2007-07-01	6000	1	6000

Как видно из таблицы, господин Иванов Иван Иванович неоднократно делал покупки. Каждый раз в таблицу добавлялся его адрес, город и телефон. А теперь представьте себе ситуацию, когда господин Иванов Иван Иванович сменил номер телефона. Каждую запись о покупке пришлось бы изменить. Кроме того, имеет место напрасная трата пространства на жестком диске. По этой причине необходимо вынести данные о клиенте в отдельную таблицу (табл. 4.2).

Таблица 4.2. *Данные о клиентах*

id_Customer	Name	Address	City	Phone
1	Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45
2	Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Теперь первоначальная табл. 4.1 примет вид табл. 4.3.

Таблица 4.3. *Таблица заказов*

id_Customer	Tovar	Date_orders	Price	Col	Sum
1	HDD	2007-06-20	3400	1	3400
2	Тюнер	2007-06-20	3100	1	3100
1	Монитор	2007-06-25	7200	1	7200

Таблица 4.3 *(окончание)*

id_Customer	Tovar	Date_orders	Price	Col	Sum
1	Тюнер	2007-06-30	3100	1	3100
1	Дискета	2007-07-01	10	10	100
1	Сканер	2007-07-01	6000	1	6000

Поле id_Customer в табл. 4.2 называется *первичным ключом* и содержит только уникальные записи. Поле id_Customer в табл. 4.3 называется *внешним ключом* и может содержать повторяющиеся записи.

Название города также можно вынести в отдельную таблицу (табл. 4.4):

Таблица 4.4. *Названия городов*

id_City	City
1	Санкт-Петербург
2	Москва

В итоге табл. 4.2 примет вид табл. 4.5.

Таблица 4.5. *Данные о клиентах*

id_Customer	Name	Address	id_City	Phone
1	Иванов Иван Иванович	Седова, 7	1	125-14-45
2	Петров Сергей Николаевич	Невский, 88	1	312-12-51

Теперь то же самое можно сделать с названиями товаров (табл. 4.6).

Таблица 4.6. *Информация о товарах*

id_Tovar	Tovar	Price
1	HDD	3400
2	Тюнер	3100
3	Монитор	7200
4	Дискета	10
5	Сканер	6000

И табл. 4.1 еще уменьшится (табл. 4.7).

Таблица 4.7. *Таблица заказов*

id_Customer	id_Tovar	Date_orders	Col	Sum
1	1	2007-06-20	1	3400
2	2	2007-06-20	1	3100
1	3	2007-06-25	1	7200
1	2	2007-06-30	1	3100
1	4	2007-07-01	10	100
1	5	2007-07-01	1	6000

Но это еще не все. Создадим еще табл. 4.8, содержащую элементы заказа.

Таблица 4.8. *Элементы заказа*

id_Orders	id_Tovar	Col
1	1	1
2	2	1
3	3	1
4	2	1
5	4	10
5	5	1

В итоге табл. 4.1 примет вид табл. 4.9.

Таблица 4.9. *Таблица заказов после нормализации*

id_Orders	id_Customer	Date_orders	Sum
1	1	2007-06-20	3400
2	2	2007-06-20	3100
3	1	2007-06-25	7200
4	1	2007-06-30	3100
5	1	2007-07-01	6100

Обратите внимание, в табл. 4.8 первичный ключ является *составным* (поля id_Orders и id_Tovar).

Такой процесс оптимизации базы данных называется *нормализацией*. Может показаться проблематичным работать с такой базой данных. Но это не так. При изменении адреса или телефона покупателя достаточно изменить эти данные только в одной таблице. А отсутствие повторяющихся записей позволит снизить размер базы данных. О том, как получить данные сразу из нескольких таблиц, мы узнаем при изучении языка SQL. Но вначале следует изучить типы данных, которые могут храниться в полях таблицы.

4.2. Типы данных полей

При создании любой таблицы необходимо принимать решение, какой тип данных будет содержать поле, одно поле может содержать данные только одного типа. Каждый из типов данных использует различный объем памяти. При выборе типа данных следует выбрать тип, который требует меньшего объема памяти.

4.2.1. Числовые типы

Числовые типы даных:

- `TINYINT[(<Длина в символах>)]` — целые числа от –128 до 127 или от 0 до 255. Занимает 1 байт;
- `BOOL` или `BOOLEAN` — либо 0, либо 1. Синоним для `TINYINT(1)`. Занимает 1 байт;
- `SMALLINT[(<Длина в символах>)]` — целые числа от –32 768 до 32 767 или от 0 до 65 535. Занимает 2 байта;
- `MEDIUMINT[(<Длина в символах>)]` — целые числа от –8 388 608 до 8 388 607 или от 0 до 16 777 215. Занимает 3 байта;
- `INT[(<Длина в символах>)]` — целое 4-байтовое число;
- `INTEGER[(<Длина в символах>)]` — синоним для `INT`;
- `BIGINT[(<Длина в символах>)]` — целое 8-байтовое число;
- `FLOAT[(<Длина в символах>, <Количество знаков после запятой>)]` — вещественные числа ±1.175494351E–38 ±3.402823466E+38. Занимает 4 байта;
- `DOUBLE[(<Длина в символах>, <Количество знаков после запятой>)]` — вещественные числа двойной точности. Занимает 8 байтов;
- `REAL` — синоним для `DOUBLE`;

- DECIMAL — дробное число, хранящееся в виде строки;
- NUMERIC — синоним для DECIMAL.

Если после типа указано слово UNSIGNED, то это означает, что поле может содержать только числа без знака.

4.2.2. Строковые типы

Текстовые типы:

- CHAR(<Длина строки>) [BINARY] — строки фиксированной длины до 255 символов. Строки будут дополняться пробелами до максимальной длины, независимо от размеров строки;
- VARCHAR(<Длина строки>) [BINARY] — строки переменной длины до 65 535 символов (до версии 5.0.3 только до 255 символов).

  Текстовый тип можно превратить в бинарный, указав модификатор BINARY;
- TINYTEXT — строка до 255 символов;
- TEXT — строка до 65 535 символов;
- MEDIUMTEXT — строка до 16 777 215 символов;
- LONGTEXT — строка до 4 294 967 295 символов.

При поиске в текстовых полях регистр символов не учитывается.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В одной таблице нельзя смешивать поля типа CHAR и VARCHAR.

Бинарные типы:

- TINYBLOB — строка до 255 символов;
- BLOB — строка до 65 535 символов;
- MEDIUMBLOB — строка до 16 777 215 символов;
- LONGBLOB — строка до 4 294 967 295 символов.

При поиске в бинарных полях учитывается регистр символов.

Перечисления и множества:

- SET('Значение1', 'Значение2', ...) — поле может содержать несколько значений из перечисленных. Может быть указано до 64 значений;
- ENUM('Значение1', 'Значение2', ...) — поле может содержать лишь одно из перечисленных значений или NULL. Может быть указано до 65 535 значений.

4.2.3. Дата и время

Календарные типы:

- `DATE` — дата в формате `ГГГГ-ММ-ДД`;
- `TIME` — время в формате `ЧЧ:ММ:СС`;
- `DATETIME` — дата и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;
- `YEAR[(2|4)]` — год в двух- или четырехсимвольном формате;
- `TIMESTAMP` — дата и время в формате timestamp. От '1970-01-01 00:00:00' до 2037 г.

4.3. Основы языка SQL

Для выборки записей из базы данных разработан специализированный язык — SQL (Structured Query Language, структурированный язык запросов). С помощью этого языка можно создавать базы данных и таблицы, добавлять, изменять и удалять данные, получать данные по запросу. В настоящее время существует множество разновидностей языка SQL. В этой главе книги мы будем изучать SQL применительно к базам данных MySQL. Обратите внимание, что некоторые SQL-команды работают только в MySQL.

4.3.1. Создание базы данных

Для создания базы данных используется команда:

```
CREATE DATABASE <Имя базы данных>;
```

Например:

```
CREATE DATABASE testDB;
```

При создании базы данных можно сразу выбрать кодировку:

```
CREATE DATABASE testDB DEFAULT CHARACTER SET cp1251 COLLATE
cp1251_general_ci;
```

Чтобы посмотреть все доступные базы данных, можно воспользоваться SQL-командой:

```
SHOW DATABASES;
```

Выбрать базу данных по умолчанию позволяет SQL-команда:

```
USE <Имя базы данных>;
```

Для тестирования команд SQL можно воспользоваться программой phpMyAdmin, которая должна быть доступна по адресу **http://localhost/Tools/phpMyAdmin/**. Итак, открываем программу. В левой части сверху находим

значок **SQL**. Если навести курсор, то появится подсказка **Окно запроса**. Щелкнем левой кнопкой мыши на значке. Откроется новое окно **Выполнить SQL запрос(ы) на БД**. В текстовом поле набираем команду:

```
CREATE DATABASE testDB DEFAULT CHARACTER SET cp1251
COLLATE cp1251_general_ci;
```

Нажимаем кнопку **Пошел**. В итоге отобразится надпись: **Ваш SQL-запрос был успешно выполнен (Запрос занял 0.0006 сек)**. Закрываем все окна, кроме первого. Для того чтобы новая база данных отобразилась в ниспадающем списке **Базы Данных**, необходимо обновить страницу. После обновления из списка выбираем пункт **testDB**. В правой части окна отобразится содержимое базы данных testDB, точнее сказать, надпись **В БД не обнаружено таблиц**, т. к. таблицы мы еще не создавали.

В верхней части страницы расположены вкладки: **Структура**, **SQL**, **Экспорт**, **Искать**, **Запрос по примеру**, **Операции** и **Уничтожить**. В дальнейшем нас будет интересовать вкладка **SQL**. Все последующие SQL-запросы мы будем набирать именно здесь.

4.3.2. Создание пользователя базы данных

После создания базы данных необходимо создать пользователя базы данных и назначить ему полномочия. *Полномочия* (или привилегии) — это права конкретного пользователя выполнять определенные действия над определенным объектом. Пользователь должен обладать наименьшим набором привилегий, необходимых для выполнения конкретных задач.

Создание и назначение полномочий осуществляется SQL-командой:

```
GRANT <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
TO <Имя пользователя> [IDENTIFIED BY '<Пароль>']
[WITH GRANT OPTION];
```

В параметре `<Привилегии>` могут быть указаны через запятую следующие полномочия:

- `ALL` или `ALL PRIVILEGES` — все полномочия;
- `USAGE` — без всех полномочий;
- `SELECT` — выбирать записи в таблицах;
- `INSERT` — вставлять новые записи в таблицы;
- `UPDATE` — изменять значения в существующих полях таблиц;
- `DELETE` — удалять записи;

- FILE — сохранять данные из таблиц в файл и восстанавливать их из файла;
- CREATE — создавать новые базы данных или таблицы. Если в команде GRANT указана определенная база данных или таблица, то пользователь может создавать только указанную базу данных или таблицу;
- ALTER — изменять структуру существующих таблиц;
- INDEX — создавать и удалять индексы определенных таблиц;
- DROP — удалить базы данных или таблицы;
- PROCESS — просматривать и удалять процессы на сервере;
- RELOAD — перезагружать таблицы полномочий;
- SHUTDOWN — останавливать сервер MySQL.

В необязательном параметре <Столбцы> может быть указан список имен столбцов, разделенных запятыми, к которым применяются привилегии.

В параметре <База данных>.<Таблица> может быть указано:

- *.* или * — полномочия предоставляются для всех баз данных в целом;
- <Имя базы данных>.* — полномочия для всех таблиц указанной базы данных;
- <Имя базы данных>.<Имя таблицы> — полномочия только для указанной таблицы в заданной базе данных. Если дополнительно указан параметр <Столбцы>, то полномочия назначаются для указанных столбцов.

В параметре <Имя пользователя> указывается имя пользователя (например, den) или комбинация имени пользователя и имени хоста (например, den@localhost). Новому пользователю можно назначить пароль.

Если указана опция WITH GRANT OPTION, то пользователь может предоставлять свои полномочия другим.

Создадим нового пользователя с именем den и назначим ему ограниченные привилегии. Для этого на вкладке **SQL** введем следующую команду:

```
GRANT select, insert, update, delete, index, alter, create, drop
ON testDB.*
TO den@localhost IDENTIFIED BY '123';
```

Нажимаем кнопку **Пошел**. В итоге отобразится надпись: **Ваш SQL-запрос был успешно выполнен (Запрос занял 0.0003 сек)**.

После создания пользователя или изменения привилегий необходимо перезагрузить привилегии с помощью SQL-команды:

```
FLUSH PRIVILEGES;
```

Для лишения пользователя полномочий используется SQL-команда:

```
REVOKE <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
TO <Имя пользователя>;
```

Если полномочия были предоставлены опцией `WITH GRANT OPTION`, то удалить их можно с помощью SQL-команды:

```
REVOKE GRANT OPTION
ON <База данных>.<Таблица>
TO <Имя пользователя>;
```

Для удаления пользователя используется SQL-команда:

```
DROP USER <Имя пользователя>;
```

Для просмотра прав пользователя предназначена SQL-команда:

```
SHOW GRANTS FOR '<Имя пользователя>'@'<Хост>';
```

Для примера выведем полномочия созданного пользователя den:

```
SHOW GRANTS FOR 'den'@'localhost';
```

4.3.3. Создание таблицы

Создать таблицу в базе данных позволяет SQL-команда:

```
CREATE TABLE <Имя таблицы> (
<Имя поля1> <Тип данных> [<Опции>],
<Имя поля2> <Тип данных> [<Опции>],
 ...
) [<Дополнительные опции>];
```

В параметре `<Опции>` могут быть указаны следующие значения:

- `NOT NULL` — поле обязательно должно иметь значение при вставке новой записи в таблицу (если не задано значение по умолчанию). Если опция не указана, то поле может быть пустым;
- `PRIMARY KEY` — поле является первичным ключом таблицы. Записи в поле должны быть уникальными. Опция может быть указана после перечисления всех полей;
- `AUTO_INCREMENT` — если при вставке новой записи оставить это поле пустым, то MySQL автоматически генерирует значение на единицу больше максимального значения, уже существующего в поле. В таблице может быть только одно поле с этой опцией;
- `DEFAULT` — значение поля по умолчанию;

- CHARACTER SET — кодировка текстового поля;
- COLLATE — тип сортировки текстового поля.

В параметре <Дополнительные опции> могут быть указаны следующие значения:

- ENGINE — тип таблицы (например, MyISAM);
- DEFAULT CHARSET — кодировка (например, cp1251);
- AUTO_INCREMENT — начальное значение для автоматической генерации значения поля.

Для вывода всех типов таблиц, поддерживаемых текущей версией MySQL, предназначена SQL-команда:

```
SHOW ENGINES;
```

На практике обычно используются два типа таблиц — MyISAM и InnoDB. Тип MyISAM является "родным" типом таблиц и применяется по умолчанию. Хотя версии MySQL под Windows по умолчанию могут устанавливать тип InnoDB. В отличие от типа MyISAM таблицы типа InnoDB поддерживают транзакции и внешние ключи, но не имеют поддержки полнотекстового поиска. Кроме того, таблицы типа InnoDB работают медленнее таблиц MyISAM, но зато они более надежны.

Для вывода всех кодировок применяется SQL-команда:

```
SHOW CHARACTER SET;
```

Чтобы получить список всех типов сортировки, можно воспользоваться SQL-командой:

```
SHOW COLLATION;
```

Создадим таблицы из нашего первоначально рассмотренного примера. Для этого в левой части из списка выбираем базу **testDB**. С правой стороны выбираем вкладку **SQL**. В текстовом поле набираем следующие команды:

```
CREATE TABLE City (
   id_City int NOT NULL auto_increment,
   City char(50) NOT NULL,
   PRIMARY KEY  (id_City)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE Customers (
   id_Customer int NOT NULL auto_increment,
   Name char(50) NOT NULL,
   Address char(255) NOT NULL,
```

```
   id_City int NOT NULL,
   Phone char(30),
   PRIMARY KEY  (id_Customer)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE Tovar (
   id_Tovar int NOT NULL auto_increment,
   Tovar char(50) NOT NULL,
   Price int NOT NULL,
   PRIMARY KEY  (id_Tovar)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE Orders_Items (
   id_Orders int NOT NULL,
   id_Tovar int NOT NULL,
   Col tinyint unsigned,
   PRIMARY KEY  (id_Orders, id_Tovar)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE Orders (
   id_Orders int NOT NULL auto_increment,
   id_Customer int NOT NULL,
   Date_orders date,
   Sum int,
   PRIMARY KEY  (id_Orders)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;
```

Чтобы выполнить запрос, нажимаем кнопку **Пошел**. Все созданные таблицы отображаются слева под списком баз данных:

```
testDB
  City
  Customers
  Orders
  Orders_Items
  Tovar
```

Если щелкнуть на названии таблицы, то справа отобразится ее структура.

Вывести все таблицы из указанной базы данных позволяет SQL-команда:

```
SHOW TABLES FROM <Имя базы данных>;
```

Для примера выведем все таблицы из базы данных testDB:

```
SHOW TABLES FROM testDB;
```

Чтобы отобразить структуру конкретной таблицы из указанной базы данных, можно воспользоваться SQL-командой:

```
SHOW COLUMNS FROM <Таблица> FROM <Имя базы данных>;
```

Для примера выведем структуру таблицы `City` из базы данных testDB:

```
SHOW COLUMNS FROM City FROM testDB;
```

Отобразить структуру таблицы позволяет также SQL-команда:

```
DESCRIBE <Таблица>;
```

В отличие от предыдущей SQL-команды перед использованием команды `DESCRIBE` база данных должна быть предварительно выбрана. Для примера выведем структуру таблицы `City` из базы данных testDB:

```
USE testDB;
DESCRIBE City;
```

4.3.4. Вставка данных в таблицу

Для добавления записей в таблицу используется SQL-команда:

```
INSERT INTO <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
```

Например, добавить две записи в таблицу `City` можно одним из следующих способов:

```
INSERT INTO City (id_City, City)
VALUES (NULL, 'Санкт-Петербург');
INSERT INTO City (id_City, City)
VALUES (NULL, 'Москва');

INSERT INTO City
SET id_City=NULL, City='Санкт-Петербург';
INSERT INTO City
SET id_City=NULL, City='Москва';

INSERT INTO City VALUES
(NULL, 'Санкт-Петербург'),
(NULL, 'Москва');
```

```
INSERT INTO City VALUES (NULL, 'Санкт-Петербург');
INSERT INTO City VALUES (NULL, 'Москва');
```

Чаще всего на практике используются последние два способа.

Для первого поля мы указали значение `NULL`, т. к. для этого поля установлена опция `auto_increment`, и MySQL автоматически вставит значение в поле.

Давайте теперь заполним наши созданные таблицы значениями. Для этого выполним следующие SQL-команды:

```
INSERT INTO City VALUES
(1, 'Санкт-Петербург'),
(2, 'Москва');

INSERT INTO Customers VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-45'),
(2, 'Петров Сергей Николаевич', 'Невский, 88', 1, '312-12-51');

INSERT INTO Tovar VALUES
(1, 'HDD', 3400),
(2, 'Тюнер', 3100),
(3, 'Монитор', 7200),
(4, 'Дискета', 10),
(5, 'Сканер', 6000);

INSERT INTO Orders_Items VALUES
(1, 1, 1),
(2, 2, 1),
(3, 3, 1),
(4, 2, 1),
(5, 4, 10),
(5, 5, 1);

INSERT INTO Orders VALUES
(1, 1, '2007-06-20', 3400),
(2, 2, '2007-06-20', 3100),
(3, 1, '2007-06-25', 7200),
(4, 1, '2007-06-30', 3100),
(5, 1, '2007-07-01', 6100);
```

Обратите внимание, что числа в кавычки не заключаются. А чтобы сохранить целостность базы данных, индексы указываются явным образом.

Если предпринимается попытка вставить запись, а в таблице уже есть запись с таким же значением первичного ключа (или значение индекса UNIQUE не уникально), то SQL-команда вставки приводит к ошибке. Если необходимо, чтобы такого рода не уникальные записи обновлялись без вывода сообщения об ошибке, можно использовать следующую SQL-команду:

```
REPLACE [INTO] <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
```

В качестве примера, изменим номер телефона господина Иванова:

```
REPLACE Customers VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-47');
```

Если передать уникальное значение, то SQL-команда REPLACE аналогична команде INSERT. Например, следующая SQL-команда добавит нового покупателя:

```
REPLACE Customers VALUES
(NULL, 'Сидоров Олег Николаевич', 'Передовиков, 12', 1, '529-15-63');
```

4.3.5. Обновление записей

Обновление записи осуществляется SQL-командой:

```
UPDATE <Имя таблицы>
SET <Поле1>='<Значение>', <Поле2>='<Значение2>', ...
WHERE <Условие>;
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Если не указано <Условие>, то будут обновлены все записи в таблице.

В параметре <Условие> могут быть указаны следующие операторы:

- = — проверка на равенство;
- > — больше;
- < — меньше;
- >= — больше или равно;
- <= — меньше или равно;
- != или <> — не равно;
- IS NOT NULL — проверка на значение;

- `IS NULL` — проверка поля на отсутствие значения;
- `BETWEEN <Начало> AND <Конец>` — проверяет, является ли значение большим или равным `<Начало>` и меньшим или равным `<Конец>`, например: `pole between 0 and 100`;
- `IN` — содержится в определенном наборе, например: `pole in ('Монитор', 'HDD')`;
- `NOT IN` — не содержится в определенном наборе, например: `pole not in ('Монитор', 'HDD')`;
- `LIKE` — соответствует шаблону SQL;
- `NOT LIKE` — не соответствует шаблону SQL.

В шаблоне SQL могут использоваться следующие символы:

- `%` — любое количество символов;
- `_` — любой одиночный символ.

Можно проверять сразу несколько условий, указав логические операции:

- `AND` — логическое И;
- `OR` — логическое ИЛИ.

Для примера, изменим телефон одного из клиентов, например, Иванова:

```
UPDATE Customers SET Phone='125-14-46' WHERE id_Customer=1;
```

Господин Иванов у нас числится под номером 1 в таблице `Customers`. Это условие мы и указали.

4.3.6. Удаление записей из таблицы

Удаление записи осуществляется SQL-командой:

```
DELETE FROM <Имя таблицы> WHERE <Условие> [ LIMIT <Число> ];
```

> ***Обратите внимание***
>
> Если условие не указано, то будут удалены все записи из таблицы.

Конструкцию `LIMIT` можно использовать для ограничения максимального количества удаляемых записей. В качестве примера удалим клиента Сидорова:

```
DELETE FROM Customers WHERE Name LIKE 'Сидоров %' LIMIT 1;
```

Для очистки определенной таблицы используется SQL-команда:

```
TRUNCATE TABLE <Имя таблицы>;
```

Частое обновление и удаление записей приводит к дефрагментации таблицы. Чтобы освободить неиспользуемое свободное пространство в таблицах типа MyISAM, можно воспользоваться SQL-командой:

```
OPTIMIZE TABLE <Имя таблицы>;
```

4.3.7. Изменение свойств таблицы

В ряде случаев нужно изменить структуру уже созданной таблицы. Для этого используется SQL-команда:

```
ALTER TABLE <Имя таблицы>
<Преобразование>;
```

В параметре <Преобразование> могут быть указаны следующие инструкции:

- RENAME <Новое имя таблицы> — переименовывает таблицу;
- ADD <Имя нового поля> <Тип данных> [FIRST | AFTER <Имя поля>] — добавляет в таблицу новое поле. Если указана опция FIRST, то поле будет добавлено в самое начало, а если AFTER <Имя поля>, то после указанного поля. По умолчанию новое поле вставляется в конец таблицы. В новом поле нужно задать значение по умолчанию, или значение NULL должно быть допустимым, т. к. в таблице уже есть записи;
- ADD PRIMARY KEY (<Имя поля>) — делает указанное поле первичным ключом;
- DROP PRIMARY KEY — удаляет первичный ключ;
- CHANGE <Имя поля> <Новое имя поля> <Новые параметры поля> — изменяет свойства столбца. С помощью этой инструкции поле можно переименовать;
- MODIFY <Имя поля> <Тип данных> — изменяет свойства столбца;
- DROP <Имя поля> — удаляет поле.

Для примера, изменим тип данных поля Address в таблице Customers:

```
ALTER TABLE Customers CHANGE Address Address char(100) NOT NULL;
```

4.3.8. Выбор записей

Выполнить запрос позволяет SQL-команда:

```
SELECT <Поле1>, <Поле2>, ...
FROM <Имя таблицы>
[ WHERE <Условие1> ]
[ GROUP BY <Имя поля1> ] [ HAVING <Условие2> ]
[ ORDER BY <Имя поля2> [desc]]
[ LIMIT <Начало>, <Количество записей> ]
```

SQL-команда SELECT ищет все записи в таблице <Имя таблицы>, которые удовлетворяют выражению <Условие1>. Если конструкция WHERE <Условие1> не указана, то будут возвращены все записи из таблицы <Имя таблицы>. Вместо перечисления полей можно указать символ *. В этом случае будут возвращены все поля.

Найденные записи при указанной конструкции ORDER BY <Имя поля2> сортируются по возрастанию. Если в конце указано слово desc, то записи будут отсортированы в обратном порядке.

Выберем все записи из таблицы City, но только из одного поля:

```
SELECT City FROM City;
```

В результате будут возвращены только названия городов:

```
Санкт-Петербург
Москва
```

Если вместо названия поля указать символ *, то будут возвращены все поля:

```
SELECT * FROM City;

1 Санкт-Петербург
2 Москва
```

Теперь выведем названия городов по алфавиту:

```
SELECT * FROM City ORDER BY City;

2 Москва
1 Санкт-Петербург
```

А теперь выведем только город с индексом 2:

```
SELECT * FROM City WHERE id_City=2;

2 Москва
```

SQL-команда SELECT позволяет выбирать записи сразу из нескольких таблиц одновременно. Для этого нужно перечислить все таблицы через запятую в конструкции FROM. В конструкции WHERE через запятую указываются пары полей, определяющие связи таблиц. Причем в условии и перечислении полей вначале указывается имя таблицы, а затем через точку имя поля.

Выведем табл. 4.2, но вместо индекса города укажем его название:

```
SELECT Customers.Name, Customers.Address, City.City, Customers.Phone
FROM Customers, City
WHERE Customers.id_City=City.id_City;
```

В итоге мы получим табл. 4.10.

Таблица 4.10. *Данные о клиентах*

Name	Address	City	Phone
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Вместо названия таблицы можно использовать псевдоним. Псевдоним создается через ключевое слово `AS` после имени таблицы в конструкции `FROM`. Перепишем предыдущий пример с использованием псевдонимов:

```
SELECT c.Name, c.Address, ct.City, c.Phone
FROM Customers AS c, City AS ct
WHERE c.id_City=ct.id_City;
```

Результат будет таким же. Кроме того, если поля в таблицах имеют разные названия, то имя таблицы можно не указывать:

```
SELECT Name, Address, City, Phone
FROM Customers AS c, City AS ct
WHERE c.id_City=ct.id_City;
```

А теперь выведем нашу первоначальную таблицу:

```
SELECT c.Name, c.Address, ct.City, c.Phone, t.Tovar, o.Date_orders,
t.Price, oi.Col
FROM Customers AS c, City AS ct, Tovar AS t, Orders AS o,
Orders_Items AS oi
WHERE c.id_City=ct.id_City AND
oi.id_Orders= o.id_Orders AND
t.id_Tovar=oi.id_Tovar AND
o.id_Customer= c.id_Customer
ORDER BY o.id_Orders;
```

В итоге мы получим табл. 4.11.

Таблица 4.11. *Таблица заказов*

Name	Address	City	Phone	Tovar	Date_orders	Price	Col
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	HDD	2007-06-20	3400	1

Таблица 4.11 *(окончание)*

Name	Address	City	Phone	Tovar	Date_orders	Price	Col
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2007-06-20	3100	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Монитор	2007-06-25	7200	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Тюнер	2007-06-30	3100	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Дискета	2007-07-01	10	10
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Сканер	2007-07-01	6000	1

Практически такая же таблица, но с исключениями:

- нет поля `Sum`. Получить это поле не так уж и сложно. Достаточно перемножить значение поля `Price` и значение поля `Col`;
- номер телефона господина Иванова изменился, т. к. мы его чуть раньше сами поменяли.

Если требуется, чтобы при поиске выдавались не все найденные записи, а определенная группа, то нужно использовать параметр `LIMIT`. Этот параметр удобен при выводе большого количества записей. Например, есть каталог из 2000 записей. Вместо того чтобы выводить его за один раз, можно выводить его частями, скажем, по 25 записей за раз. В этом параметре задается два значения: `<Начало>` и `<Количество записей>`:

```
SELECT * FROM City ORDER BY City LIMIT 0, 25;
```

Обратите внимание, первая запись имеет индекс 0. Если бы в таблице `City` было бы более 25 записей, то мы бы получили только первые 25.

Кроме всего, команда `SELECT` позволяет использовать следующие функции:

- `COUNT(<Поле>)` — количество ненулевых значений в указанном поле;
- `MIN(<Поле>)` — минимальное значение в указанном поле;
- `MAX(<Поле>)` — максимальное значение в указанном поле;
- `SUM(<Поле>)` — сумма значений в указанном поле;
- `AVG(<Поле>)` — средняя величина значений в указанном поле.

Выведем общее количество заказов:

```
SELECT COUNT(id_Orders) FROM Orders;
```

Вывод: 5. Теперь минимальную сумму заказа:

```
SELECT MIN(Sum) FROM Orders;
```

Вывод: 3100. А теперь максимальную сумму заказа:

```
SELECT MAX(Sum) FROM Orders;
```

Вывод: 7200.

Для получения более подробной информации можно воспользоваться конструкцией `GROUP BY`. Например, можно посмотреть среднюю сумму покупок каждого покупателя:

```
SELECT id_Customer, AVG(Sum) AS s
FROM Orders
GROUP BY id_Customer
ORDER BY s;
```

Мы используем псевдоним для имени поля и по нему сортируем записи от меньшего результата к большему.

Если нужно выбрать клиентов, заказавших больше определенной суммы, то можно воспользоваться конструкцией `HAVING`. Она выполняет те же функции, что и конструкция `WHERE`, но только для конструкции `GROUP BY`:

```
SELECT id_Customer, AVG(Sum) AS s
FROM Orders
GROUP BY id_Customer
HAVING s > 4000
ORDER BY s;
```

Для определения эффективности SQL-запроса используется оператор `EXPLAIN`. Оператор имеет следующий формат:

```
EXPLAIN <Имя таблицы>;
EXPLAIN <Запрос SELECT>;
```

Первый вариант выведет структуру указанной таблицы, а второй вариант позволяет выяснить, каким образом выполняется запрос с помощью SQL-команды `SELECT`. В качестве примера, выведем результат поиска клиента по его полному имени. SQL-команду будем выполнять с помощью программы MySQL monitor:

```
EXPLAIN SELECT * FROM Customers WHERE Name='Иванов Иван Иванович'\G
```

Вывод:

```
table: Customers
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 2
Extra: Using where
```

Рассмотрим результат выполнения оператора `EXPLAIN`. Строка `table` содержит название таблицы. Значение строки `type` показывает эффективность выполнения запроса. Может принимать значения `ALL`, `index`, `range`, `ref`, `eq_ref`, `const` (или `system`). Все перечисленные значения расположены по возрастанию степени эффективности запроса. `ALL` означает, что просматриваются все записи таблицы. Это не самый эффективный способ. Количество просматриваемых записей указывается в строке `rows`. Чем меньше это число, тем эффективнее запрос. Строка `possible_keys` содержит список всех доступных ключей или значение `NULL` в случае отсутствия ключей. В строке `key` указано название используемого индекса, а строка `key_len` содержит длину используемого ключа. Поля, которые дополнительно использовались для выборки, указываются в строке `ref`. Последняя строка `Extra` обычно содержит дополнительную информацию о выполнении запроса.

Как видно из примера, для выполнения запроса пришлось просматривать все записи таблицы `Customers`, т. к. записи в неиндексированных полях таблицы расположены в произвольном порядке. Для ускорения выполнения запросов применяются *индексы* (ключи). Индексированные поля всегда поддерживаются в отсортированном состоянии, что позволяет быстро найти необходимую запись, не просматривая все записи. Неиндексированное поле можно сравнить с текстом этой книги, а индексированное поле — с предметным указателем. Чтобы найти описание какой-либо функции, в первом случае необходимо последовательно перелистывать страницы книги. Во втором случае достаточно найти функцию по алфавиту в предметном указателе, а затем сразу перейти на указанную страницу. Необходимо заметить, что применение индексов приводит к увеличению размера базы данных, а также к затратам времени на поддержание индекса в отсортированном состоянии при каждом запросе. По этой причине индексировать следует поля, которые очень часто используются в запросах типа:

```
SELECT <Список полей> FROM <Таблица> WHERE <Поле>=<Значение>;
```

Существуют следующие виды индексов:

- первичный ключ;
- уникальный индекс;
- обычный индекс;
- индекс FULLTEXT.

Первичный ключ служит для однозначной идентификации каждой записи в таблице. Обратите внимание, в качестве первичного ключа может служить только одно поле в таблице. Для создания индекса используется ключевое слово PRIMARY KEY. При создании таблицы ключевое слово можно указать после определения параметров поля:

```
CREATE TABLE City (
   id_City int NOT NULL PRIMARY KEY auto_increment,
   City char(50) NOT NULL
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;
```

Или после перечисления всех полей:

```
CREATE TABLE City (
   id_City int NOT NULL auto_increment,
   City char(50) NOT NULL,
   PRIMARY KEY (id_City)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;
```

Добавить первичный ключ в существующую таблицу позволяет SQL-команда:

```
ALTER TABLE <Таблица> ADD PRIMARY KEY (<Поле>);
```

Кроме того, первичный ключ можно создать на нескольких столбцах:

```
PRIMARY KEY (id_Orders, id_Tovar)
```

Удалить первичный ключ позволяет SQL-команда:

```
ALTER TABLE <Таблица> DROP PRIMARY KEY;
```

Обычных и уникальных индексов в таблице может быть несколько. Создать индекс можно при определении структуры таблицы с помощью ключевых слов INDEX и KEY (UNIQUE INDEX и UNIQUE KEY для уникального индекса):

```
CREATE TABLE Customers (
   id_Customer int NOT NULL auto_increment,
   Name char(50) NOT NULL,
   Address char(255) NOT NULL,
```

```
   id_City int NOT NULL,
   Phone char(30),
   PRIMARY KEY (id_Customer),
   KEY MyIndex (Name)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;
```

Индекс может иметь название. Но т. к. название индекса не указывается в SQL-запросе, то чаще всего названием индекса служит имя поля. Сервер MySQL самостоятельно решает, каким индексом лучше воспользоваться в каждой конкретной ситуации. Знать название индекса необходимо для его удаления из таблицы.

При индексировании текстовых полей следует указать количество символов (до 1000 символов), подлежащих индексации:

```
KEY MyIndex (Name(10))
```

Создать обычный индекс позволяет SQL-команда:

```
CREATE INDEX <Имя индекса> ON <Таблица> (<Поле>(<Количество символов>));
```

Или:

```
ALTER TABLE <Таблица>
ADD INDEX <Имя индекса> (<Поле>(<Количество символов>));
```

Создать уникальный индекс позволяет SQL-команда:

```
CREATE UNIQUE INDEX <Имя индекса>
ON <Таблица> (<Поле>(<Количество символов>));
```

Или:

```
ALTER TABLE <Таблица>
ADD UNIQUE INDEX <Имя индекса> (<Поле>(<Количество символов>));
```

Удалить обычный и уникальный индексы позволяет SQL-команда:

```
DROP INDEX <Имя индекса> ON <Таблица>;
```

Или:

```
ALTER TABLE <Таблица> DROP INDEX <Имя индекса>;
```

В качестве примера создадим индекс для поля `Name` таблицы `Customers`:

```
CREATE INDEX Name ON Customers (Name(50));
```

А теперь сделаем запрос и проверим его эффективность с помощью оператора `EXPLAIN`:

```
EXPLAIN SELECT * FROM Customers WHERE Name='Иванов Иван Иванович'\G
```

Вывод:

```
table: Customers
type: ref
possible_keys: Name
key: Name
key_len: 50
ref: const
rows: 1
Extra: Using where
```

Сравните результат запроса с индексом и предыдущий пример без индекса. Обратите внимание, значение строки `type` уже не равно `ALL`, а количество просмотренных записей равно 1. Это означает, что индекс полностью задействован.

Индекс `FULLTEXT` применяется для полнотекстового поиска. Реализацию полнотекстового поиска и способы создания индекса `FULLTEXT` мы подробно рассмотрим в *разд. 4.7*.

Получить полную информацию об индексах таблицы позволяет SQL-команда:

```
SHOW INDEX FROM <Таблица> [FROM <База данных>];
```

Например:

```
SHOW INDEX FROM Customers\G
```

Вывод:

```
*************************** 1. row ***************************
Table: Customers
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: id_Customer
Collation: A
Cardinality: 2
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
*************************** 2. row ***************************
```

```
Table: Customers
Non_unique: 1
Key_name: Name
Seq_in_index: 1
Column_name: Name
Collation: A
Cardinality: NULL
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
```

Строка `Collation` показывает способ сортировки (`A` — по возрастанию, `D` — по убыванию, `NULL` — без сортировки), строка `Cardinality` позволяет узнать количество элементов в индексе, а строка `Index_type` информирует о методе индексации.

В качестве значения строки `Cardinality` для индекса `Name` мы получили значение `NULL`. Может показаться, что в индексе нет элементов. Чтобы получить количество элементов, необходимо перед использованием оператора `SHOW INDEX` выполнить SQL-команду:

```
ANALYZE TABLE <Таблица>;
```

4.3.9. Удаление таблицы и базы данных

Удалить таблицу позволяет SQL-команда:

```
DROP TABLE <Имя таблицы>;
```

Удалить всю базу данных позволяет SQL-команда:

```
DROP DATABASE <Имя базы данных>;
```

4.4. Операторы MySQL

Операторы позволяют произвести определенные действия с данными. Например, математические операторы позволяют произвести арифметические вычисления. Рассмотрим операторы, доступные в MySQL.

> ***Обратите внимание***
>
> Выполнять SQL-команды мы будем в программе MySQL monitor. Способы запуска утилиты мы рассматривали в *разд. 4.1*.

4.4.1. Математические операторы

Математические операторы:

- `+` — сложение:

```
SELECT 8 + 5;
```

- `-` — вычитание:

```
SELECT 10 — 5;
```

- `*` — умножение:

```
SELECT 10 * 5;
```

- `/` — деление:

```
SELECT 10 / 5;
```

  Вывод: `2.0000`;

- `DIV` — целочисленное деление:

```
SELECT 10 DIV 5;
```

  Вывод: `2;`

```
SELECT 10 DIV 3;
```

  Вывод: `3`;

- `%` — остаток от деления:

```
SELECT 10 % 2;
```

  Вывод: `0;`

```
SELECT 9 % 2;
```

  Вывод: `1`;

- `MOD` — остаток от деления:

```
SELECT 10 MOD 2;
```

  Вывод: `0`.

Вместо операторов `%` и `MOD` можно использовать функцию `MOD()`.

```
SELECT MOD(10, 2);
```

Вывод: `0`.

Следует отметить, что если один из операндов равен `NULL`, то результат операции также будет равен `NULL`. В отличие от языков программирования деление на нуль не приводит к генерации сообщения об ошибке. Результатом операции деления на нуль является значение `NULL`.

Если необходимо сменить знак, то перед операндом следует указать знак `-`.

```
SELECT -(-5);
```

Вывод: 5.

В качестве примера рассмотрим возможность подсчета переходов по рекламной ссылке. Для этого создадим таблицу `counter` в базе данных testDB:

```
CREATE TABLE counter (
   id_link int NOT NULL auto_increment,
   total int,
   PRIMARY KEY (id_link)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим одну запись:

```
INSERT INTO counter VALUES (1, 0);
```

Для подсчета переходов в тексте ссылки укажем идентификатор в базе данных и URL-адрес:

```
<HTML>
<HEAD>
<TITLE>Подсчет переходов по ссылкам</TITLE>
</HEAD>
<BODY>
<A href="cgi-bin/go.pl?id=1&url=http://www.mail.ru/">Перейти</A>
</BODY>
</HTML>
```

Регистрация переходов производится в файле go.pl. Исходный код файла приведен в листинге 4.1.

Листинг 4.1. Регистрация переходов по ссылке

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
use DBI;
use CGI qw( :standard );

my $id = param('id') || 0;
my $url = param('url') || "";

if ($url ne "" && $id =~ /^[0-9]+$/ && $id != 0) {
   # Подключаемся к базе данных
   my $ds = 'DBI:mysql:testDB:localhost';
   my $db = DBI->connect($ds, 'den', '123');
```

```
    if ($db) {
        my $query = 'UPDATE counter SET total = total + 1 ';
        $query .= "WHERE id_link=$id";
        $db->do($query); # Выполняем запрос
        $db->disconnect(); # Закрываем соединение
    }
    print "Location: $url\n\n";
}
else {
    print "Content-type: text/html\n\n";
    print "Ошибка";
}
```

Использование оператора + позволяет произвести увеличение счетчика за один запрос. Иначе пришлось бы вначале получить значение из базы данных, затем увеличить значение и только во втором запросе обновить значение в базе данных.

4.4.2. Двоичные операторы

Двоичные операторы:

- ~ — двоичная инверсия;
- & — двоичное И;
- | — двоичное ИЛИ;
- ^ — двоичное исключающее ИЛИ;
- << — сдвиг влево — сдвиг влево на один или более разряд с заполнением младших разрядов нулями;
- >> — сдвиг вправо — сдвиг вправо на один или более разряд с заполнением старших разрядов содержимым самого старшего разряда.

4.4.3. Операторы сравнения

Операторы сравнения используются для создания запросов в конструкциях WHERE и HAVING. Перечислим их:

- = — равно;
- <=> — эквивалентно;
- != — не равно;

- <> — не равно;
- < — меньше;
- > — больше;
- <= — меньше или равно;
- >= — больше или равно;
- IS NOT NULL — проверка на значение;
- IS NULL — проверка поля на отсутствие значения;
- BETWEEN <Начало> AND <Конец> — проверяет, является ли значение большим или равным <Начало> и меньшим или равным <Конец>, например: pole between 0 and 100;
- IN — содержится в определенном наборе, например, pole in ('Монитор', 'HDD');
- NOT IN — не содержится в определенном наборе, например: pole not in ('Монитор', 'HDD');
- LIKE — соответствует шаблону SQL;
- NOT LIKE — не соответствует шаблону SQL;
- RLIKE — соответствует регулярному выражению;
- REGEXP — соответствует регулярному выражению (синоним RLIKE);
- NOT REGEXP — не соответствует регулярному выражению (синоним NOT RLIKE);
- NOT RLIKE — не соответствует регулярному выражению.

В шаблоне SQL могут использоваться следующие символы:

- % — любое количество символов;
- _ — любой одиночный символ.

Можно проверять сразу несколько условий, указав логические операции:

- AND — логическое И;
- OR — логическое ИЛИ;
- XOR — исключающее логическое ИЛИ.

Результатом операции сравнения являются:

- 0 — ложь;
- 1 — истина;
- NULL.

Исключением является оператор эквивалентности <=>. Он возвращает только два значения: 0 (ложь) и 1 (истина). Этот оператор введен специально для сравнения значения NULL.

Следует отметить, что по умолчанию сравнение строк происходит без учета регистра. Если указать ключевое слово BINARY, то регистр символов будет учитываться:

```
SELECT 'TEXT'='text';
```

Вывод: 1 (истина).

```
SELECT BINARY 'TEXT'='text';
```

Вывод: 0 (ложь).

Результат сравнения можно изменить на противоположный с помощью операторов ! и NOT:

```
SELECT 'TEXT'='text';
```

Вывод: 1 (истина).

```
SELECT !('TEXT'='text');
```

Вывод: 0 (ложь).

```
SELECT NOT ('TEXT'='text');
```

Вывод: 0 (ложь).

Логическое выражение следует заключить в круглые скобки, т. к. приоритет оператора отрицания выше приоритета других операторов.

4.4.4. Приоритет выполнения операторов

При составлении выражений следует учитывать приоритет выполнения операторов. Перечислим операторы в порядке убывания приоритета:

- BINARY, COLLATE;
- NOT, !;
- - (унарный минус), ~;
- *, /, %, MOD, DIV;
- +, – — сложение, вычитание;
- <<, >> — двоичные сдвиги;
- & — двоичное И;
- | — двоичное ИЛИ;
- =, <=>, >=, <=, >, <, <>, !=, IS, LIKE, REGEXP, IN;
- BETWEEN;

- `&&`, `AND`;
- `||`, `OR`, `XOR`.

С помощью круглых скобок можно изменить последовательность выполнения выражения:

```
SELECT 5 + 3 * 7;
```

Вывод: 26.

```
SELECT (5 + 3) * 7;
```

Вывод: 56.

4.4.5. Преобразование типов данных

В большинстве случаев преобразование типов осуществляется автоматически. В этом разделе мы рассмотрим результаты автоматического преобразования типов, а также встроенные функции для специального приведения типов.

Что будет, если к числу прибавить строку?

```
SELECT '5' + 3;
```

Вывод: 8.

```
SELECT '5st' + 3;
```

Вывод: 8.

В этом случае строка преобразуется в число, а затем выполняется операция сложения. Но что будет, если строку невозможно преобразовать в число?

```
SELECT 'str' + 3;
```

Вывод: 3.

```
SELECT 3 + 'str';
```

Вывод: 3.

Если строку невозможно преобразовать в число, то она приравнивается к нулю.

Для преобразования типов используются две функции:

- `CAST(<Выражение> AS <Тип>)`;
- `CONVERT(<Выражение>, <Тип>)`.

Параметр `<Тип>` может принимать следующие значения:

- `BINARY`;
- `CHAR`;

- DATE;
- DATETIME;
- SIGNED [INTEGER];
- TIME;
- UNSIGNED [INTEGER].

4.5. Поиск по шаблону

Для поиска по шаблону используются два оператора:

- LIKE — соответствует шаблону SQL;
- NOT LIKE — не соответствует шаблону SQL.

В шаблоне SQL могут использоваться следующие специальные символы:

- % — любое количество символов;
- _ — любой одиночный символ.

Если спецсимволы не используются, то операторы LIKE и = эквивалентны:

```
SELECT 'строка для поиска' LIKE 'поиск';
```

Вывод: 0.

```
SELECT 'поиск' LIKE 'поиск';
```

Вывод: 1.

Следует учитывать, что поиск в текстовом поле производится без учета регистра. Чтобы учитывался регистр, необходимо указать ключевое слово BINARY:

```
SELECT 'Perl' LIKE 'perl';
```

Вывод: 1.

```
SELECT 'Perl' LIKE BINARY 'perl';
```

Вывод: 0.

Специальные символы могут быть расположены в любом месте шаблона. Например, чтобы найти все вхождения, необходимо указать символ % в начале и конце шаблона:

```
SELECT 'строка для поиска' LIKE '%поиск%';
```

Вывод: 1.

Можно установить привязку или только к началу строки, или только к концу:

```
SELECT 'строка для поиска' LIKE 'строка%';
```

Вывод: 1.

```
SELECT 'новая строка для поиска' LIKE 'строка%';
```

Вывод: 0.

```
SELECT 'строка для поиска' LIKE '%поиска';
```

Вывод: 1.

```
SELECT 'строка для поиска 2' LIKE '%поиска';
```

Вывод: 0.

Шаблон для поиска может иметь очень сложную структуру:

```
SELECT 'строка для поиска' LIKE '%строк_%по_ск%';
```

Вывод: 1.

```
SELECT 'строка для поиска' LIKE '%поиск%а';
```

Вывод: 1.

Обратите внимание на последнюю строку поиска. Этот пример демонстрирует, что спецсимвол `%` соответствует не только любому количеству символов, но и полному их отсутствию.

Что же делать, если необходимо найти символы `%` и `_`? Ведь они являются специальными:

```
SELECT 'скидка 10%' LIKE '%10%';
```

Вывод: 1.

```
SELECT 'скидка 10$' LIKE '%10%';
```

Вывод: 1.

В этом случае специальные символы необходимо экранировать с помощью обратного слэша:

```
SELECT 'скидка 10%' LIKE '%10\%';
```

Вывод: 1.

```
SELECT 'скидка 10$' LIKE '%10\%';
```

Вывод: 0.

Метод `quote()` из модуля `DBI` не добавляет защитный слэш перед символами `%` и `_`. Следует добавить его самостоятельно с помощью оператора `s///`.

```
$text =~ s/%/\\%/g;
$text =~ s/_/\\_/g;
```

В качестве примера рассмотрим поиск по шаблону. Для этого создадим таблицу `search` в базе данных testDB:

```
CREATE TABLE search (
   id int NOT NULL auto_increment,
   str text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим две записи:

```
INSERT INTO search VALUES (NULL, 'Скидка 10%');
INSERT INTO search VALUES (NULL, 'Скидка 10$');
```

Исходный код с вариантами поиска по шаблону приведен в листинге 4.2.

Листинг 4.2. Поиск по шаблону

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
use DBI;
print "Content-type: text/html; charset=windows-1251\n\n";

my $str_search = '10%';
# Подключаемся к базе данных
my $ds = 'DBI:mysql:testDB:localhost';
my $db = DBI->connect($ds, 'den', '123');
if (!$db) {
   print "Не удалось установить соединение с базой данных";
   exit();
}
$db->do("SET NAMES cp1251");
# Добавляем защитные слэши
$str_search =~ s/\\/\\\\/g;
$str_search =~ s/'/\\'/g;

print "Без добавления слэшей перед спецсимволами:<BR>";
my $query = "SELECT str FROM search WHERE str LIKE '%";
$query .= $str_search;
$query .= "%'";
my $res = $db->prepare($query);
$res->execute();
```

```
if (!$res->err) {
   while (my @row = $res->fetchrow_array()) {
      print "$row[0]<BR>\n";
   }
}
$res->finish();

print "<BR>После добавления слэшей перед спецсимволами:<BR>";
$str_search =~ s/%/\\%/g;
$str_search =~ s/_/\\_/g;
$query = "SELECT str FROM search WHERE str LIKE '%";
$query .= $str_search;
$query .= "%'";
my $result = $db->prepare($query);
$result->execute();
if (!$result->err) {
   while (my @pole = $result->fetchrow_array()) {
      print "$pole[0]<BR>\n";
   }
}
$result->finish();
$db->disconnect(); # Закрываем соединение
```

Вывод:

```
Без добавления слэшей перед спецсимволами:
Скидка 10%
Скидка 10$

После добавления слэшей перед спецсимволами:
Скидка 10%
```

В этом примере предполагается, что значение переменной `$str_search` было получено через форму поиска. Поэтому, прежде чем подставить значение переменной в SQL-запрос, мы экранируем специальные символы. Иначе любой пользователь может видоизменить SQL-запрос.

4.6. Поиск с помощью регулярных выражений

Регулярные выражения позволяют осуществить сложный поиск. Использовать регулярные выражения позволяют следующие операторы:

- `RLIKE` — соответствует регулярному выражению;
- `REGEXP` — соответствует регулярному выражению (синоним `RLIKE`);
- `NOT REGEXP` — не соответствует регулярному выражению (синоним `NOT RLIKE`);
- `NOT RLIKE` — не соответствует регулярному выражению.

При использовании регулярных выражений следует помнить, что поиск выполняется более медленно, чем при поиске по шаблону, и требует больше ресурсов сервера.

Метасимволы, используемые в регулярных выражениях:

- `^` — привязка к началу строки;
- `$` — привязка к концу строки:

  ```
  SELECT '2' RLIKE '^[0-9]+$';
  ```

  Вывод: `1`.

  ```
  SELECT 'Строка2' RLIKE '^[0-9]+$';
  ```

  Вывод: `0`.

  Если убрать привязку к началу и концу строки, то любая строка, содержащая число, вернет `1`:

  ```
  SELECT 'Строка2' RLIKE '[0-9]+';
  ```

  Вывод: `1`.

  Можно указать привязку только к началу или только к концу строки:

  ```
  SELECT 'Строка2' RLIKE '[0-9]+$';
  ```

  Вывод: `1`.

  ```
  SELECT 'Строка2' RLIKE '^[0-9]+';
  ```

  Вывод: `0`.

  Регулярное выражение `'^$'` соответствует пустой строке. Чтобы найти значение `NULL`, необходимо использовать не регулярные выражения, а операторы `IS NULL` и `IS NOT NULL`;
- `[[:<:]]` — привязка к началу слова;

- `[[:>:]]` — привязка к концу слова:

```
SET NAMES 'cp1251';
SELECT 'в середине строки' RLIKE '[[:<:]]середине[[:>:]]';
```

Вывод: `1`.

При работе с русским языком необходимо правильно настроить кодировку. Если кодировка соединения будет `latin1`, то поиск вернет значение 0, а не 1;

- `[]` — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире:
  - `[09]` — число 0 или 9;
  - `[0-9]` — число от 0 до 9;
  - `[абв]` — буквы "а", "б" и "в";
  - `[а-г]` — буквы "а", "б", "в" и "г";
  - `[а-я]` — буквы от "а" до "я";
  - `[0-9а-яа-z]` — любая цифра и любая буква независимо от языка.

  Значение можно инвертировать, если после первой скобки указать символ `^`. В итоге можно указать символы, которых не должно быть на этом месте в строке:
  - `[^09]` — не число 0 или 9;
  - `[^0-9]` — не число от 0 до 9;
  - `[^a-яа-z]` — не буква.

  Поиск производится без учета регистра символов. Чтобы учитывался регистр, необходимо указать ключевое слово `BINARY`:

```
SELECT 'String' RLIKE '^[a-z]+$';
```

Вывод: `1`.

```
SELECT 'String' RLIKE BINARY '^[a-z]+$';
```

Вывод: `0`.

Следует заметить, что учет регистра русских букв зависит от настройки кодировки:

```
SET NAMES 'cp866';
SELECT 'Строка' RLIKE '^[а-я]+$';
```

Вывод: `1`.

```
SET NAMES 'cp1251';
SELECT 'Строка' RLIKE '^[а-я]+$';
```

Вывод: `0`.

```
SET NAMES 'latin1';
SELECT 'Строка' RLIKE '^[а-я]+$';
```

Вывод: `0`.

Как видно из примера, с кодировкой cp1251 также возникла проблема. По этой причине для русских букв стоит учитывать регистр:

```
SET NAMES 'cp1251';
SELECT 'Строка' RLIKE '^[А-Яа-я]+$';
```

Вывод: `1`.

Вместо перечисления символов можно использовать стандартные классы:

- `[[:alnum:]]` — алфавитно-цифровые символы;
- `[[:alpha:]]` — буквенные символы;
- `[[:lower:]]` — строчные буквы;
- `[[:upper:]]` — прописные буквы;
- `[[:digit:]]` — десятичные цифры;
- `[[:xdigit:]]` — шестнадцатеричные цифры;
- `[[:punct:]]` — знаки пунктуации;
- `[[:blank:]]` — символы табуляции и пробелов;
- `[[:space:]]` — символы пробела, табуляции, новой строки или возврата каретки;
- `[[:cntrl:]]` — управляющие символы;
- `[[:print:]]` — печатные символы;
- `[[:graph:]]` — печатные символы, за исключением пробельных;
- . (точка) — любой символ.

Что же делать, если нужно найти точку, ведь символ "точка" соответствует любому символу? Для этого перед специальным символом необходимо указать два символа \ (\\.):

```
SELECT '26,03.2008' RLIKE
'^[0-3][[:digit:]].[01][[:digit:]].[12][09][[:digit:]][[:digit:]]$';
```

Вывод: `1`.

```
SELECT '26,03.2008' RLIKE
'^[0-3][[:digit:]]\\.[01][[:digit:]]\\.[12][09][[:digit:]][[:digit:]]$';
```

Вывод: `0` (перед точкой указаны символы "\\").

Количество вхождений символа в строку задается с помощью *квантификаторов*:

- `{n}` — соответствует `n` вхождениям символа в строку:

  `[[:digit:]]{2}` — соответствует двум вхождениям любой цифры;

- `{n,}` — по крайней мере, `n` вхождений символа в строку или более:

  `[[:digit:]]{2,}` — соответствует двум и более вхождениям любой цифры;

- `{n,m}` — не менее `n` вхождений символа в строку и не более `m`. Цифры указываются через запятую без пробела:

  `[[:digit:]]{2,5}` — соответствует от двух до пяти вхождениям любой цифры;

- `*` — ноль или большее число вхождений символа в строку:

  `[[:digit:]]*` — цифры могут не встретиться в строке или встретиться много раз;

- `+` — одно или больше вхождений символа в строку:

  `[[:digit:]]+` — цифра может встретиться один раз или встретиться много раз;

- `?` — ноль или одно вхождение символа в строку:

  `[[:digit:]]?` — цифра может встретиться один раз или не встретиться совсем.

Для указания количества вхождений нескольких символов используются круглые скобки:

```
SELECT '121212' RLIKE '^(12){3}$';
```

Вывод: `1`.

Логическое ИЛИ:

`n|m` — соответствует одному из символов `n` или `m`:

`красн(ая)|(ое)` — `красная` или `красное`, но не `красный`:

```
SELECT 'красная' RLIKE 'красн(ая)|(ое)';
```

Вывод: `1`.

4.7. Режим полнотекстового поиска

Кроме поиска по шаблону и применения регулярных выражений, для таблиц типа MyISAM можно использовать режим полнотекстового поиска. Столбцы, используемые для поиска, должны быть проиндексированы при помощи специального индекса FULLTEXT. Индексации подлежат столбцы, имеющие тип CHAR, VARCHAR или TEXT.

4.7.1. Создание индекса *FULLTEXT*

Создать индекс FULLTEXT можно следующими способами:

- при создании таблицы с помощью оператора CREATE TABLE:

```
FULLTEXT INDEX <Название индекса> (<Столбцы через запятую>)

CREATE TABLE search1 (
   id int NOT NULL auto_increment,
   str text,
   FULLTEXT INDEX index1 (str),
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

CREATE TABLE search2 (
   id int NOT NULL auto_increment,
   str1 text,
   str2 text,
   FULLTEXT INDEX index2 (str1, str2),
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

- при помощи оператора ALTER TABLE:

```
CREATE TABLE search3 (
   id int NOT NULL auto_increment,
   str text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

ALTER TABLE search3 ADD FULLTEXT index3 (str);
```

```
CREATE TABLE search4 (
   id int NOT NULL auto_increment,
   str1 text,
   str2 text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

ALTER TABLE search4 ADD FULLTEXT index4 (str1, str2);
```

- при помощи оператора `CREATE INDEX`:

```
CREATE TABLE search5 (
   id int NOT NULL auto_increment,
   str text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

CREATE FULLTEXT INDEX index5 ON search5 (str);

CREATE TABLE search6 (
   id int NOT NULL auto_increment,
   str1 text,
   str2 text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;

CREATE FULLTEXT INDEX index6 ON search6 (str1, str2);
```

В индекс попадут слова длиной от 4 до 84 символов. Данные значения задаются переменными `ft_min_word_len` и `ft_max_word_len` соответственно. Изменить значения этих переменных можно через конфигурационный файл my.cnf. После изменения значения переменных необходимо заново создать индекс `FULLTEXT`. Посмотреть текущие значения переменных позволяет SQL-команда:

```
SHOW VARIABLES LIKE 'ft%';
```

Следует отметить, что полнотекстовый поиск создавался для поиска в большом объеме текста. Если поле состоит из нескольких слов, то оно может вообще не попасть в индекс.

4.7.2. Реализация полнотекстового поиска

Полнотекстовый поиск выполняется с помощью конструкции `MATCH(...) AGAINST(...)`. Конструкция имеет следующий формат:

```
MATCH(<Поля через запятую>)
AGAINST('<Строка для поиска>' [<Модификатор>])
```

Необязательный параметр `<Модификатор>` может принимать следующие значения:

- `IN BOOLEAN MODE` — режим логического поиска;
- `WITH QUERY EXPANSION` — поиск с расширением запроса.

Для примера добавим три записи в таблицу `search1`:

```
INSERT INTO search1 VALUES (NULL, 'В индекс попадут слова длиной от 4
до 84 символов. Данные значения задаются переменными ft_min_word_len и
ft_max_word_len соответственно. Изменить значения этих переменных можно
через конфигурационный файл my.cnf. После изменения значения переменных
необходимо заново создать индексы FULLTEXT.');
INSERT INTO search1 VALUES (NULL, 'Запись 2');
INSERT INTO search1 VALUES (NULL, 'Строка 3');
```

> ***Обратите внимание***
>
> Для реализации полнотекстового поиска в таблице должно быть не менее трех записей.

А теперь найдем строку с помощью полнотекстового поиска:

```
SELECT * FROM search1 WHERE MATCH(str) AGAINST('значения');
```

Результаты поиска сортируются по коэффициенту релевантности. Коэффициент представляет собой число с плавающей точкой. Чтобы увидеть этот коэффициент для разных запросов, воспользуемся следующими SQL-запросами:

```
SELECT MATCH(str) AGAINST('конфигурационный файл') AS iq
FROM search1 WHERE MATCH(str) AGAINST('конфигурационный файл');
```

Вывод: `0.99839779903687`.

```
SELECT MATCH(str) AGAINST('конфигурационный файлы') AS iq
FROM search1 WHERE MATCH(str) AGAINST('конфигурационный файлы');
```

Вывод: `0.49919889951843`.

4.7.3. Режим логического поиска

Режим логического поиска позволяет использовать специальные символы, которые влияют на значение коэффициента релевантности. Чтобы применить режим логического поиска, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `IN BOOLEAN MODE`. Перечислим специальные символы логического режима:

- `+` — слово обязательно должно присутствовать в результате:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('конфигурационный +файл' IN BOOLEAN MODE);
```

- `-` — слово не должно присутствовать в результате:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('конфигурационный -файл' IN BOOLEAN MODE);
```

- `<` — уменьшает вклад слова в коэффициент релевантности:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('конфигурационный <файл' IN BOOLEAN MODE);
```

- `>` — увеличивает вклад слова в коэффициент релевантности:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('конфигурационный >файл' IN BOOLEAN MODE);
```

- `()` — круглые скобки служат для группировки слов в подвыражения;
- `~` — символ для указания нежелательного слова. Символ `~` (в отличие от символа `-`) не исключает слово из результата, а лишь уменьшает коэффициент релевантности;
- `*` — символ усечения. Указывается в конце слова;
- `""` — строка должна содержать точную фразу:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('"конфигурационный файл"' IN BOOLEAN MODE);
```

Если в логическом режиме не используются специальные символы, то коэффициент релевантности для всех строк будет одинаковым.

4.7.4. Поиск с расширением запроса

При поиске с расширением запроса происходит двойной поиск. Сначала будут найдены искомые слова, а затем слова из результатов поиска. Чтобы применить режим поиска с расширением запроса, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `WITH QUERY EXPANSION`:

```
SELECT * FROM search1
WHERE MATCH(str)
AGAINST('конфигурационный файл' WITH QUERY EXPANSION);
```

4.8. Функции MySQL

MySQL имеет множество встроенных функций. Например, функция `NOW()` возвращает текущую дату и время, а функция `DATE_FORMAT()` преобразует формат вывода даты.

При использовании функций следует помнить, что между круглыми скобками и именем функции не должно быть пробела, а скобки следует обязательно указывать, даже если в функцию не передаются аргументы.

Рассмотрим функции MySQL более подробно.

4.8.1. Функции для работы с числами

Стандартные тригонометрические функции:

- `SIN()` — синус;
- `COS()` — косинус;
- `TAN()` — тангенс;
- `COT()` — котангенс.

Обратные тригонометрические функции:

- `ASIN()` — арксинус;
- `ACOS()` — арккосинус;
- `ATAN()` — арктангенс.

Округление чисел:

- `CEILING()` — значение, округленное до ближайшего большего целого:

  ```
  SELECT CEILING(4.3);
  ```

  Вывод: `5`;

- `CEIL()` — значение, округленное до ближайшего большего целого:

  ```
  SELECT CEIL(4.3);
  ```

  Вывод: `5`;

- `FLOOR()` — значение, округленное до ближайшего меньшего целого:

  ```
  SELECT FLOOR(4.6);
  ```

  Вывод: `4`;

- `ROUND()` — значение, округленное до ближайшего меньшего целого, для чисел с дробной частью меньше 0,5 или значение, округленное до ближайшего большего целого, для чисел с дробной частью больше 0,5. Если дробная часть числа равна 0,5, то округление зависит от версии MySQL. Начиная с версии 5.0.3, округление производится в большую сторону:

  ```
  SELECT ROUND(4.49);
  ```

  Вывод: `4`.

  ```
  SELECT ROUND(4.5);
  ```

  Вывод: `4 или 5`. Это зависит от версии MySQL.

  ```
  SELECT ROUND(4.501);
  ```

  Вывод: `5`.

  ```
  SELECT ROUND(4.51);
  ```

  Вывод: `5`;

- `TRUNCATE(X, Y)` — возвращает дробное число `X` с `Y` количеством знаков после запятой. Если в качестве значения аргумента `Y` передать значение 0, то функция вернет число, округленное до меньшего целого:

  ```
  SELECT TRUNCATE(4.55, 0);
  ```

  Вывод: `4`.

  ```
  SELECT TRUNCATE(4.55, 1);
  ```

  Вывод: `4.5`.

  ```
  SELECT TRUNCATE(4.55, 3);
  ```

  Вывод: `4.550`;

Функции для преобразования чисел:

- `CONV(<Число>, <Исходная система>, <Нужная система>)` — преобразует число из одной системы счисления в другую:

  ```
  SELECT CONV(255, 10, 16);
  ```

  Вывод: `FF`.

  ```
  SELECT CONV('FF', 16, 10);
  ```

  Вывод: `255`;

- `BIN(<Число>)` — преобразует число из десятичной системы счисления в двоичную:

  ```
  SELECT BIN(17);
  ```

  Вывод: `10001`;

- `HEX()` — возвращает значение аргумента в виде шестнадцатеричного числа:

  ```
  SELECT HEX(255);
  ```

  Вывод: `FF`;

- `OCT(<Число>)` — преобразует число из десятичной системы счисления в восьмеричную:

  ```
  SELECT OCT(10);
  ```

  Вывод: `12`.

Прочие функции:

- `ABS()` — абсолютное значение:

  ```
  SELECT ABS(-4.55);
  ```

  Вывод: `4.55`;

- `EXP()` — экспонента;
- `LOG(X)` — натуральный логарифм;
- `LOG2(X)` — логарифм числа по основанию 2:

  ```
  SELECT LOG2(128);
  ```

  Вывод: `7`;

- `LOG10(X)` — логарифм числа по основанию 10:

  ```
  SELECT LOG10(100);
  ```

  Вывод: `2`;

- `LOG(<Основание>, X)` — логарифм числа `X` по основанию `<Основание>`;

  ```
  SELECT LOG(2, 128);
  ```

  Вывод: `7`.

  ```
  SELECT LOG(10, 100);
  ```

  Вывод: `2`;

- `POW(<Число>, <Степень>)` — возводит `<Число>` в `<Степень>`:

  ```
  SELECT POW(5, 2);
  ```

  Вывод: `25`;

- `SQRT()` — квадратный корень:

  ```
  SELECT SQRT(25);
  ```

  Вывод: `5`;

- `PI()` — возвращает число π:

  ```
  SELECT PI();
  ```

  Вывод: `3.141593`;

- `MOD(X, Y)` — остаток от деления `X` на `Y`:

  ```
  SELECT MOD(10, 2);
  ```

  Вывод: `0`;

- `DEGREES()` — значение угла, преобразованное из радиан в градусы:

  ```
  SELECT DEGREES(PI());
  ```

  Вывод: `180`;

- `RADIANS()` — значение угла, преобразованное из градусов в радианы;

  ```
  SELECT RADIANS(180);
  ```

  Вывод: `3.1415926535898`;

- `SIGN()` — возвращает –1, если число отрицательное; 1, если число положительное; 0, если число равно нулю:

  ```
  SELECT SIGN(-80);
  ```

  Вывод: `-1`.

  ```
  SELECT SIGN(80);
  ```

  Вывод: `1`;

- `LEAST()` — возвращает минимальное значение из списка:

  ```
  SELECT LEAST(1, 2, 3);
  ```

  Вывод: `1`;

- `GREATEST()` — возвращает максимальное значение из списка:

  ```
  SELECT GREATEST(1, 2, 3);
  ```

  Вывод: `3`;

- `FORMAT(<Число>, <Количество знаков после запятой>)` — возвращает строку, в которой число выводится с определенным количеством знаков после запятой, а каждые три разряда отделяются запятой:

  ```
  SELECT FORMAT(56873.8732, 2);
  ```

  Вывод: `56,873.87`;

- `RAND()` — возвращает случайное число в диапазоне от 0 до 1. Если в функцию передать параметр, то это настроит генератор на новую последовательность. Следует учитывать, что один и тот же параметр выдаст одну и ту же последовательность при следующем вызове функции:

  ```
  SELECT RAND();
  ```

  Вывод: `0.48179545857179`.

```
SELECT RAND();
```

Вывод: `0.54826072281996`.

```
SELECT RAND(10);
```

Вывод: `0.65705152196535`.

```
SELECT RAND(10);
```

Вывод: `0.65705152196535`.

В качестве примера рассмотрим вывод записи из базы данных случайным образом. Предположим, что у нас развлекательный портал и есть таблица в базе данных, заполненная анекдотами. При каждом запросе страницы мы будем выводить один анекдот случайным образом. Для этого в базе данных testDB создадим таблицу `anecdotes`:

```
CREATE TABLE anecdotes (
   id int NOT NULL auto_increment,
   anecdote text,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим несколько записей:

```
INSERT INTO anecdotes VALUES (NULL, 'Анекдот 1');
INSERT INTO anecdotes VALUES (NULL, 'Анекдот 2');
INSERT INTO anecdotes VALUES (NULL, 'Анекдот 3');
INSERT INTO anecdotes VALUES (NULL, 'Анекдот 4');
INSERT INTO anecdotes VALUES (NULL, 'Анекдот 5');
```

Исходный код для вывода анекдота случайным образом приведен в листинге 4.3.

Листинг 4.3. Вывод анекдота случайным образом

```
#!/usr/bin/perl -w
use CGI::Carp qw(fatalsToBrowser);
use strict;
use DBI;
print "Content-type: text/html; charset=windows-1251\n\n";

# Подключаемся к базе данных
my $ds = 'DBI:mysql:testDB:localhost';
my $db = DBI->connect($ds, 'den', '123');
```

```
if (!$db) {
   print "Не удалось установить соединение с базой данных";
   exit();
}
$db->do("SET NAMES cp1251");

my $query = "SELECT * FROM anecdotes ORDER BY RAND() LIMIT 1";
my $res = $db->prepare($query);
$res->execute();
if (!$res->err) {
   my @row = $res->fetchrow_array();
   print "$row[1]<BR>\n";
}
$res->finish();
$db->disconnect(); # Закрываем соединение
```

4.8.2. Функции даты и времени

Функции для получения текущей даты и времени:

- `NOW()` — в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:
  `SELECT NOW();`
  Вывод: `2008-09-21 22:35:04`;
- `LOCALTIME()` — в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:
  `SELECT LOCALTIME();`
  Вывод: `2008-09-21 22:35:04`;
- `LOCALTIMESTAMP()` — в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:
  `SELECT LOCALTIMESTAMP();`
  Вывод: `2008-09-21 22:35:04`;
- `UTC_TIMESTAMP()` — по Гринвичу в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:
  `SELECT UTC_TIMESTAMP();`
  Вывод: `2008-09-21 18:36:21`;
- `SYSDATE()` — возвращает текущие дату и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:
  `SELECT SYSDATE();`
  Вывод: `2008-09-21 22:36:43`;

- `CURDATE()` — возвращает текущую дату в формате `ГГГГ-ММ-ДД`:

  ```
  SELECT CURDATE();
  ```

  Вывод: `2008-09-21`;

- `CURRENT_DATE()` — возвращает текущую дату в формате `ГГГГ-ММ-ДД`:

  ```
  SELECT CURRENT_DATE();
  ```

  Вывод: `2008-09-21`;

- `UTC_DATE()` — возвращает текущую дату по Гринвичу в формате `ГГГГ-ММ-ДД`:

  ```
  SELECT UTC_DATE();
  ```

  Вывод: `2008-09-21`;

- `CURTIME()` — возвращает текущее время в формате `ЧЧ:ММ:СС`:

  ```
  SELECT CURTIME();
  ```

  Вывод: `22:38:01`;

- `CURRENT_TIME()` — возвращает текущее время в формате `ЧЧ:ММ:СС`:

  ```
  SELECT CURRENT_TIME();
  ```

  Вывод: `22:38:19`;

- `UTC_TIME()` — возвращает текущее время по Гринвичу в формате `ЧЧ:ММ:СС`:

  ```
  SELECT UTC_TIME();
  ```

  Вывод: `18:38:38`;

- `UNIX_TIMESTAMP()` — возвращает число секунд, прошедших с полуночи 1 января 1970 г.:

  ```
  SELECT UNIX_TIMESTAMP();
  ```

  Вывод: `1222022340`.

Функции для получения фрагментов даты и времени:

- `DATE()` — возвращает дату:

  ```
  SELECT DATE('2008-09-21 22:36:43');
  ```

  Вывод: `2008-09-21`;

- `YEAR()` — возвращает год:

  ```
  SELECT YEAR('2008-09-21 22:36:43');
  ```

  Вывод: `2008`;

- `MONTH()` — возвращает месяц:

  ```
  SELECT MONTH('2008-09-21 22:36:43');
  ```

  Вывод: `9`;

- MONTHNAME() — возвращает название месяца в виде строки:

```
SELECT MONTHNAME('2008-09-21 22:36:43');
```

Вывод: September;

- DAY() — возвращает номер дня в месяце:

```
SELECT DAY('2008-09-21 22:36:43');
```

Вывод: 21;

- DAYOFMONTH() — возвращает номер дня в месяце:

```
SELECT DAYOFMONTH('2008-09-21 22:36:43');
```

Вывод: 21;

- TIME() — возвращает время:

```
SELECT TIME('2008-09-21 22:36:43');
```

Вывод: 22:36:43;

- HOUR() — возвращает час:

```
SELECT HOUR('2008-09-21 22:36:43');
```

Вывод: 22;

- MINUTE() — возвращает минуты:

```
SELECT MINUTE('2008-09-21 22:36:43');
```

Вывод: 36;

- SECOND() — возвращает секунды:

```
SELECT SECOND('2008-09-21 22:36:43');
```

Вывод: 43;

- MICROSECOND() — возвращает микросекунды:

```
SELECT MICROSECOND('2008-09-21 22:36:43.123456');
```

Вывод: 123456.

Вместо перечисленных функций можно использовать функцию EXTRACT(). Функция имеет следующий формат:

```
EXTRACT(<Тип> FROM <Дата и время>)
```

Параметр <Тип> может принимать значения:

- YEAR — год:

```
SELECT EXTRACT(YEAR FROM '2008-09-21 22:36:43');
```

Вывод: 2008;

- YEAR_MONTH — год и месяц:

```
SELECT EXTRACT(YEAR_MONTH FROM '2008-09-21 22:36:43');
```

Вывод: 200809;

- MONTH — месяц:

```
SELECT EXTRACT(MONTH FROM '2008-09-21 22:36:43');
```

Вывод: 9;

- DAY — день:

```
SELECT EXTRACT(DAY FROM '2008-09-21 22:36:43');
```

Вывод: 21;

- DAY_HOUR — день и час:

```
SELECT EXTRACT(DAY_HOUR FROM '2008-09-21 22:36:43');
```

Вывод: 2122;

- DAY_MINUTE — день, час и минуты:

```
SELECT EXTRACT(DAY_MINUTE FROM '2008-09-21 22:36:43');
```

Вывод: 212236;

- DAY_SECOND — день, час, минуты и секунды:

```
SELECT EXTRACT(DAY_SECOND FROM '2008-09-21 22:36:43');
```

Вывод: 21223643;

- DAY_MICROSECOND — день, час, минуты, секунды и микросекунды:

```
SELECT EXTRACT(DAY_MICROSECOND FROM '2008-09-21 22:36:43.111111');
```

Вывод: 21223643111111;

- HOUR — час:

```
SELECT EXTRACT(HOUR FROM '2008-09-21 22:36:43');
```

Вывод: 22;

- HOUR_MINUTE — час и минуты:

```
SELECT EXTRACT(HOUR_MINUTE FROM '2008-09-21 22:36:43');
```

Вывод: 2236;

- HOUR_SECOND — час, минуты и секунды:

```
SELECT EXTRACT(HOUR_SECOND FROM '2008-09-21 22:36:43');
```

Вывод: 223643;

- HOUR_MICROSECOND — час, минуты, секунды и микросекунды:

```
SELECT EXTRACT(HOUR_MICROSECOND FROM '2008-09-21 22:36:43.111111');
```

Вывод: 223643111111;

- MINUTE — минуты:

```
SELECT EXTRACT(MINUTE FROM '2008-09-21 22:36:43');
```

Вывод: 36;

- MINUTE_SECOND — минуты и секунды:

```
SELECT EXTRACT(MINUTE_SECOND FROM '2008-09-21 22:36:43');
```

Вывод: 3643;

- MINUTE_MICROSECOND — минуты, секунды и микросекунды:

```
SELECT EXTRACT(MINUTE_MICROSECOND FROM '2008-09-21 22:36:43.111111');
```

Вывод: 3643111111;

- SECOND — секунды:

```
SELECT EXTRACT(SECOND FROM '2008-09-21 22:36:43');
```

Вывод: 43;

- SECOND_MICROSECOND — секунды и микросекунды:

```
SELECT EXTRACT(SECOND_MICROSECOND FROM '2008-09-21 22:36:43.111111');
```

Вывод: 43111111;

- MICROSECOND — микросекунды:

```
SELECT EXTRACT(MICROSECOND FROM '2008-09-21 22:36:43.111111');
```

Вывод: 111111.

Функции для получения дополнительных сведений о дате:

- QUARTER() — порядковый номер квартала в году (от 1 до 4):

```
SELECT QUARTER('2008-09-21');
```

Вывод: 3;

- WEEKOFYEAR() — порядковый номер недели в году (от 1 до 53):

```
SELECT WEEKOFYEAR('2008-09-21');
```

Вывод: 38;

- WEEK() — порядковый номер недели в году (от 0 до 53). Неделя начинается с воскресенья:

```
SELECT WEEK('2008-09-21');
```

Вывод: 38;

- YEARWEEK() — число в формате ГГГГНН, где ГГГГ — год, а НН — порядковый номер недели в году (от 0 до 53):

```
SELECT YEARWEEK('2008-09-21');
```

Вывод: 200838;

- DAYOFYEAR() — порядковый номер дня в году (от 1 до 366):

```
SELECT DAYOFYEAR('2008-09-21');
```

Вывод: 265;

- MAKEDATE(<Год>, <Номер дня в году>) — возвращает дату в формате ГГГГ-ММ-ДД по номеру дня в году:

  ```
  SELECT MAKEDATE(2008, 265);
  ```

  Вывод: 2008-09-21;

- DAYOFWEEK() — порядковый номер дня недели (1 — для воскресенья, 2 — для понедельника, ..., 7 — для субботы):

  ```
  SELECT DAYOFWEEK('2008-09-21');
  ```

  Вывод: 1;

- WEEKDAY() — порядковый номер дня недели (0 — для понедельника, 1 — для вторника, ..., 6 — для воскресенья):

  ```
  SELECT WEEKDAY('2008-09-21');
  ```

  Вывод: 6;

- DAYNAME() — название дня недели на английском языке:

  ```
  SELECT DAYNAME('2008-09-21');
  ```

  Вывод: Sunday;

- TO_DAYS(<Дата>) — количество дней, прошедших с нулевого года:

  ```
  SELECT TO_DAYS('2008-09-21');
  ```

  Вывод: 733671;

- FROM_DAYS(<Количество дней>) — возвращает дату в формате ГГГГ-ММ-ДД по количеству дней, прошедших с нулевого года:

  ```
  SELECT FROM_DAYS(733671);
  ```

  Вывод: 2008-09-21;

- TIME_TO_SEC(<Время>) — количество секунд, прошедших с начала суток:

  ```
  SELECT TIME_TO_SEC('12:52:35');
  ```

  Вывод: 46355;

- SEC_TO_TIME(<Количество секунд>) — возвращает время в формате ЧЧ:ММ:СС по количеству секунд, прошедших с начала суток:

  ```
  SELECT SEC_TO_TIME(46355);
  ```

  Вывод: 12:52:35.

Функции для манипуляции датой и временем:

- ADDDATE(<Дата>, INTERVAL <Интервал> <Тип>) — прибавляет к параметру <Дата> временной интервал;
- DATE_ADD(<Дата>, INTERVAL <Интервал> <Тип>) — прибавляет к параметру <Дата> временной интервал;

- SUBDATE(<Дата>, INTERVAL <Интервал> <Тип>) — вычитает из параметра <Дата> временной интервал;
- DATE_SUB(<Дата>, INTERVAL <Интервал> <Тип>) — вычитает из параметра <Дата> временной интервал.

Параметр <Тип> в функциях ADDDATE(), DATE_ADD(), SUBDATE() и DATE_SUB() может принимать следующие значения:

- YEAR — год;
- YEAR_MONTH — год и месяц (формат 'ГГ-ММ');
- MONTH — месяц;
- DAY — день;
- DAY_HOUR — день и час (формат 'ДД ЧЧ');
- DAY_MINUTE — день, час и минуты (формат 'ДД ЧЧ:ММ');
- DAY_SECOND — день, час, минуты и секунды (формат 'ДД ЧЧ:ММ:СС');
- DAY_MICROSECOND — формат 'ДД ЧЧ:ММ:СС.XXXXXX';
- HOUR — час;
- HOUR_MINUTE — час и минуты (формат 'ЧЧ:ММ');
- HOUR_SECOND — час, минуты и секунды (формат 'ЧЧ:ММ:СС');
- HOUR_MICROSECOND — формат 'ЧЧ:ММ:СС.XXXXXX';
- MINUTE — минуты;
- MINUTE_SECOND — минуты и секунды (формат 'ММ:СС');
- MINUTE_MICROSECOND — формат 'ММ:СС.XXXXXX';
- SECOND — секунды;
- SECOND_MICROSECOND — формат 'СС.XXXXXX';
- MICROSECOND — микросекунды.

Например:

```
SELECT ADDDATE('2008-09-21 22:36:43', INTERVAL '6 3:5:15' DAY_SECOND);
```

Вывод: 2008-09-28 01:41:58;

- операторы + и – — добавить или вычесть интервал времени можно с помощью операторов + и –:

```
SELECT '2008-09-21 22:36:43' + INTERVAL '3:7:15' HOUR_SECOND;
```

Вывод: 2008-09-22 01:43:58.

```
SELECT '2008-09-21 22:36:43' — INTERVAL '3:7:15' HOUR_SECOND;
```

Вывод: `2008-09-21 19:29:28`;

- `ADDDATE(<Дата>, <Интервал в днях>)` — прибавляет к параметру `<Дата>` временной интервал в днях. Если указать перед интервалом знак минуса, то интервал вычитается из даты:

```
SELECT ADDDATE('2008-09-21', 10);
```

Вывод: `2008-10-01`.

```
SELECT ADDDATE('2008-09-21', -10);
```

Вывод: `2008-09-11`;

- `ADDTIME(<Дата>, <Время>)` — прибавляет к параметру `<Дата>` временной интервал:

```
SELECT ADDTIME('2008-09-21 22:36:43', '12:52:35');
```

Вывод: `2008-09-22 11:29:18`;

- `SUBTIME(<Дата>, <Время>)` — вычитает из параметра `<Дата>` временной интервал:

```
SELECT SUBTIME('2008-09-21 22:36:43', '12:52:35');
```

Вывод: `2008-09-21 09:44:08`;

- `DATEDIFF(<Конечная дата>, <Начальная дата>)` — вычисляет количество дней между двумя датами:

```
SELECT DATEDIFF('2008-09-27', '2008-09-21');
```

Вывод: `6`;

- `TIMEDIFF(<Конечная дата>, <Начальная дата>)` — вычисляет разницу между двумя временными значениями:

```
SELECT TIMEDIFF('2008-09-21 22:36:43', '2008-09-21 15:36:43');
```

Вывод: `07:00:00`;

- `PERIOD_ADD(<Дата>, <Количество месяцев>)` — добавляет заданное `<Количество месяцев>` к дате, заданной в формате `ГГГГММ` или `ГГММ`:

```
SELECT PERIOD_ADD(200804, 4);
```

Вывод: `200808`;

- `PERIOD_DIFF(<Конечная дата>, <Начальная дата>)` — вычисляет разницу в месяцах между двумя временными значениями, заданными в формате `ГГГГММ` или `ГГММ`:

```
SELECT PERIOD_DIFF(200808, 200804);
```

Вывод: `4`;

- CONVERT_TZ(<Дата>, <Часовой пояс1>, <Часовой пояс2>) — переводит дату из одного часового пояса в другой:

  ```
  SELECT CONVERT_TZ('2008-09-21 22:36:43', '+00:00', '+4:00');
  ```

  Вывод: 2008-09-22 02:36:43;

- LAST_DAY(<Дата>) — возвращает дату в формате ГГГГ-ММ-ДД, в которой день выставлен на последний день текущего месяца:

  ```
  SELECT LAST_DAY('2008-09-21 22:36:43');
  ```

  Вывод: 2008-09-30;

- MAKETIME(<Часы>, <Минуты>, <Секунды>) — возвращает время в формате ЧЧ:ММ:СС:

  ```
  SELECT MAKETIME(12, 52, 35);
  ```

  Вывод: 12:52:35;

- TIMESTAMP(<Дата>, [<Время>]) — возвращает дату в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС:

  ```
  SELECT TIMESTAMP('2008-09-21');
  ```

  Вывод: 2008-09-21 00:00:00.

  ```
  SELECT TIMESTAMP('2008-09-21', '12:52:35');
  ```

  Вывод: 2008-09-21 12:52:35.

Функции для форматирования даты и времени:

- DATE_FORMAT(<Дата>, <Формат>) — форматирует дату в соответствии со строкой <Формат>:

  ```
  SELECT DATE_FORMAT('2008-09-21 22:36:43', '%d.%m.%Y');
  ```

  Вывод: 21.09.2008;

- STR_TO_DATE(<Дата>, <Формат>) — возвращает дату в форматах ГГГГ-ММ-ДД ЧЧ:ММ:СС или ГГГГ-ММ-ДД по дате соответствующей строке <Формат>:

  ```
  SELECT STR_TO_DATE('21.09.2008 12:52:35', '%d.%m.%Y %H:%i:%s');
  ```

  Вывод: 2008-09-21 12:52:35;

- TIME_FORMAT(<Время>, <Формат>) — форматирует время в соответствии со строкой <Формат>:

  ```
  SELECT TIME_FORMAT('12:52:35', '%H %i %s');
  ```

  Вывод: 12 52 35;

- FROM_UNIXTIME(<Дата>, [<Формат>]) — возвращает дату в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС или соответствующую строке <Формат> по количеству секунд, прошедших с полуночи 1 января 1970 г.:

```
SELECT FROM_UNIXTIME(1208135919);
```

Вывод: `2008-04-14 05:18:39`.

```
SELECT FROM_UNIXTIME(1208135919, '%d.%m.%Y');
```

Вывод: `14.04.2008`;

- `GET_FORMAT(<Тип времени>, '<Стандарт>')` — возвращает строку форматирования для пяти стандартов времени. Параметр `<Тип времени>` может принимать значения:
  - `DATETIME` — дата и время;
  - `DATE` — дата;
  - `TIME` — время.

  Параметр `<Стандарт>` может принимать значения:
  - `ISO` — стандарт ISO;
  - `EUR` — европейский стандарт;
  - `USA` — американский стандарт;
  - `JIS` — японский стандарт;
  - `INTERNAL` — интернациональный формат.

```
SELECT GET_FORMAT(DATE, 'EUR');
```

Вывод: `%d.%m.%Y`.

```
SELECT DATE_FORMAT('2008-09-21 22:36:43', GET_FORMAT(DATE, 'EUR'));
```

Вывод: `21.09.2008`.

Параметр `<Формат>` в функциях форматирования может содержать следующие зарезервированные комбинации символов:

- `%Y` — год из 4 цифр;
- `%y` — год из 2 цифр;
- `%m` — номер месяца с предваряющим нулем (от 01 до 12);
- `%c` — номер месяца без предваряющего нуля (от 1 до 12);
- `%b` — аббревиатура месяца из 3 букв по-английски;
- `%M` — полное название месяца по-английски;
- `%d` — номер дня с предваряющим нулем (от 01 до 31);
- `%e` — номер дня без предваряющего нуля (от 1 до 31);
- `%w` — номер дня недели (0 — для воскресенья и 6 — для субботы);
- `%a` — аббревиатура дня недели из 3 букв по-английски;

- `%W` — полное название дня недели по-английски;
- `%H` — часы в 24-часовом формате (от 00 до 23);
- `%h` — часы в 12-часовом формате (от 01 до 12);
- `%i` — минуты (от 00 до 59);
- `%s` — секунды (от 00 до 59);
- `%f` — микросекунды.

4.8.3. Функции для обработки строк

Перечислим основные функции для обработки строк:

- `CHAR_LENGTH(<Строка>)` — возвращает количество символов в строке:
  ```
  SELECT CHAR_LENGTH('String');
  ```
  Вывод: `6`.

  Функция корректно работает с многобайтовыми кодировками;

- `CHARACTER_LENGTH(<Строка>)` — возвращает количество символов в строке:
  ```
  SELECT CHARACTER_LENGTH('String');
  ```
  Вывод: `6`.

  Функция корректно работает с многобайтовыми кодировками;

- `LENGTH(<Строка>)` — возвращает количество символов в строке. `SELECT LENGTH('String');`

  Вывод: `6`.

  Функция некорректно работает с многобайтовыми кодировками, т. к. возвращает количество байтов;

- `BIT_LENGTH(<Строка>)` — возвращает длину строки в битах:
  ```
  SELECT BIT_LENGTH('String');
  ```
  Вывод: `48`;

- `TRIM([[<Откуда>] [<Символы для удаления>] FROM] <Строка>)` — удаляет из начала (и/или конца) строки символы, указанные в параметре `<Символы для удаления>`. Если параметр не указан, то удаляемыми символами являются пробелы. Необязательный параметр `<Откуда>` может принимать значения:

  - `BOTH` — символы удаляются из начала и конца строки (значение по умолчанию);
  - `LEADING` — только из начала строки;

- TRAILING — только из конца строки:

```
SELECT TRIM('   String   ');
```

Вывод: 'String'.

```
SELECT TRIM(LEADING FROM '   String   ');
```

Вывод: 'String   '.

```
SELECT TRIM(TRAILING FROM '   String   ');
```

Вывод: '   String'.

```
SELECT TRIM(BOTH 'm' FROM 'mmmmStringmmmm');
```

Вывод: 'String'.

```
SELECT TRIM(TRAILING 'ng' FROM 'String');
```

Вывод: 'Stri';

- LTRIM(<Строка>) — удаляет пробелы в начале строки:

```
SELECT LTRIM('   String   ');
```

Вывод: 'String   ';

- RTRIM(<Строка>) — удаляет пробелы в конце строки:

```
SELECT RTRIM('   String   ');
```

Вывод: '   String';

- LOWER(<Строка>) — переводит все символы в нижний регистр:

```
SELECT LOWER('STRING');
```

Вывод: string;

- LCASE(<Строка>) — переводит все символы в нижний регистр:

```
SELECT LCASE('STRING');
```

Вывод: string;

- UPPER(<Строка>) — переводит все символы в верхний регистр:

```
SELECT UPPER('string');
```

Вывод: STRING;

- UCASE(<Строка>) — переводит все символы в верхний регистр:

```
SELECT UCASE('string');
```

Вывод: STRING;

- REVERSE(<Строка>) — возвращает строку в обратном порядке:

```
SELECT REVERSE('string');
```

Вывод: gnirts;

- LEFT(<Строка>, <Количество символов>) — возвращает заданное количество крайних символов слева:

  ```
  SELECT LEFT('string', 2);
  ```

  Вывод: st;

- RIGHT(<Строка>, <Количество символов>) — возвращает заданное количество крайних символов справа:

  ```
  SELECT RIGHT('string', 2);
  ```

  Вывод: ng;

- SUBSTRING(<Строка>, <Начальная позиция>, [<Длина>]) — возвращает подстроку длиной <Длина>, начиная с позиции <Начальная позиция>. Если параметр <Длина> не задан, то возвращаются все символы до конца строки:

  ```
  SELECT SUBSTRING('string', 2, 2);
  ```

  Вывод: tr.

  ```
  SELECT SUBSTRING('string', 2);
  ```

  Вывод: tring;

- MID(<Строка>, <Начальная позиция>, [<Длина>]) — возвращает подстроку длиной <Длина> начиная с позиции <Начальная позиция>. Если параметр <Длина> не задан, то возвращаются все символы до конца строки:

  ```
  SELECT MID('string', 2, 2);
  ```

  Вывод: tr.

  ```
  SELECT MID('string', 2);
  ```

  Вывод: tring;

- LPAD(<Строка>, <Длина>, <Подстрока>) — добавляет подстроку к исходной строке слева до длины <Длина>:

  ```
  SELECT LPAD('string', 10, 'mp');
  ```

  Вывод: mpmpstring;

- RPAD(<Строка>, <Длина>, <Подстрока>) — добавляет подстроку к исходной строке справа до длины <Длина>:

  ```
  SELECT RPAD('string', 10, 'mp');
  ```

  Вывод: stringmpmp;

- CONCAT(<Строка1>, <Строка2>, ..., <СтрокаN>) — объединяет все параметры в одну строку:

  ```
  SELECT CONCAT('string1', 'string2', 'string3');
  ```

  Вывод: string1string2string3;

- `CONCAT_WS(<Разделитель>, <Строка1>, ..., <СтрокаN>)` — объединяет все параметры в одну строку через разделитель, заданный в параметре `<Разделитель>`:

  ```
  SELECT CONCAT_WS(' — ', 'string1', 'string2', 'string3');
  ```

  Вывод: `string1 — string2 — string3`;

- `REPEAT(<Строка>, <Количество повторений>)` — возвращает строку, полученную заданным количеством повторений исходной строки:

  ```
  SELECT REPEAT('str', 3);
  ```

  Вывод: `strstrstr`;

- `SPACE(<Количество пробелов>)` — возвращает строку, состоящую из заданного количества пробелов:

  ```
  SELECT CONCAT(SPACE(3), 'String');
  ```

  Вывод: `'   String'`;

- `ELT(<Номер из списка>, <Строка1>, ..., <СтрокаN>)` — возвращает строку из списка параметров, заданную номером из первого параметра:

  ```
  SELECT ELT(2, 'string1', 'string2', 'string3');
  ```

  Вывод: `string2`;

- `ASCII(<Строка>)` — возвращает ASCII-код первого символа строки:

  ```
  SELECT ASCII('String');
  ```

  Вывод: `83`;

- `ORD(<Строка>)` — возвращает ASCII-код первого символа строки. Корректно работает с многобайтовыми кодировками:

  ```
  SELECT ORD('String');
  ```

  Вывод: `83`;

- `CHAR(<ASCII-код1>, <ASCII-код2>, ..., <ASCII-кодN>)` — возвращает строку, состоящую из последовательности символов, соответствующих ASCII-кодам:

  ```
  SELECT CHAR(83, 116, 114, 105, 110, 103);
  ```

  Вывод: `String`;

- `INSTR(<Строка>, <Подстрока>)` — возвращает позицию первого вхождения подстроки в строку. Если вхождение не найдено, то возвращается 0:

  ```
  SELECT INSTR('string', 'st');
  ```

  Вывод: `1`;

- `LOCATE(<Подстрока>, <Строка>, [<Начальная позиция>])` — возвращает позицию первого вхождения подстроки в строку. Если вхождение не найде-

но, то возвращается 0. Если начальная позиция не указана, то поиск производится с начала строки:

```
SELECT LOCATE('st', 'string_st');
```

Вывод: 1.

```
SELECT LOCATE('st', 'string_st', 3);
```

Вывод: 8;

- POSITION(<Подстрока> IN <Строка>) — возвращает позицию первого вхождения подстроки в строку. Если вхождение не найдено, то возвращается 0:

```
SELECT POSITION('st' IN 'string');
```

Вывод: 1.

```
SELECT POSITION('pt' IN 'string');
```

Вывод: 0;

- FIELD(<Исходная строка>, <Строка1>, ..., <СтрокаN>) — возвращает номер строки из списка <Строка1>, ..., <СтрокаN>, которая совпадает с исходной строкой:

```
SELECT FIELD('st', 'string', 'st', 'st2');
```

Вывод: 2;

- FIND_IN_SET(<Исходная строка>, <Список строк через запятую>) — возвращает номер строки из списка <Список строк через запятую>, которая совпадает с исходной строкой:

```
SELECT FIND_IN_SET('st', 'string,st,st2');
```

Вывод: 2;

- REPLACE(<Строка>, <Подстрока для замены>, <Новая подстрока>) — производит замену всех вхождений и возвращает результат в виде новой строки:

```
SELECT REPLACE('Привет, Петя', 'Петя', 'Вася');
```

Вывод: Привет, Вася;

- SUBSTRING_INDEX(<Строка>, <Подстрока>, <Номер вхождения>) — находит заданное вхождение подстроки в строку и возвращает часть строки, расположенную слева от подстроки:

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 1);
```

Вывод: синий.

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 2);
```

Вывод: синий, красный.

Если параметр <Номер вхождения> имеет отрицательное значение, то находит заданное вхождение подстроки с конца строки и возвращает часть строки, расположенную справа от подстроки:

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', -1);
```

Вывод: `зеленый`.

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', -2);
```

Вывод: `красный, зеленый`;

- `INSERT(<Строка>, <Начальная позиция>, <Длина>, <Подстрока>)` — заменяет фрагмент в строке с начальной позиции длиной `<Длина>` на значение параметра `<Подстрока>`:

```
SELECT INSERT('красный', 6, 2, 'ое');
```

Вывод: `красное`.

```
SELECT INSERT('красный', 6, 1, 'ое');
```

Вывод: `красное`;

- `QUOTE(<Строка>)` — экранирует все специальные символы в строке:

```
SELECT QUOTE("Д'Артаньян и три мушкетера");
```

Вывод: `'Д\'Артаньян и три мушкетера'`;

- `UNHEX(<Строка>)` — каждая пара символов в строке воспринимается как шестнадцатеричное число, которое преобразуется в символ:

```
SELECT UNHEX('537472696E67');
```

Вывод: `String`;

- `COMPRESS(<Строка>)` — архивирует строку. Сжатую строку следует хранить в полях, имеющих бинарный тип данных;
- `UNCOMPRESS(<Строка>)` — разархивирует строку, сжатую функцией `COMPRESS()`;
- `UNCOMPRESSED_LENGTH(<Строка>)` — возвращает длину строки, которую она будет иметь после разархивирования:

```
SELECT UNCOMPRESSED_LENGTH(COMPRESS('Строка'));
```

Вывод: `6`;

- `CHARSET(<Строка>)` — возвращает название кодировки для строки:

```
SET NAMES 'cp866';
SELECT CHARSET('Строка');
```

Вывод: `cp866`;

- `COLLATION(<Строка>)` — возвращает порядок сортировки для строки:

```
SET NAMES 'cp866';
SELECT COLLATION('Строка');
```

Вывод: `cp866_general_ci`;

- STRCMP(<Строка1>, <Строка2>) — сравнивает две строки и возвращает:
  - 0 — если строки идентичны;
  - -1 — если <Строка1> больше <Строка2>;
  - 1 — если <Строка1> меньше <Строка2>:

  ```
  SELECT STRCMP('Строка', 'Строка');
  ```

  Вывод: 0.

  ```
  SELECT STRCMP('Строка1', 'Строка2');
  ```

  Вывод: -1.

  ```
  SELECT STRCMP('Строка2', 'Строка1');
  ```

  Вывод: 1;

- LOAD_FILE(<Путь к файлу>) — возвращает содержимое файла в виде строки. Часто используется для заполнения бинарных полей.

  В качестве примера создадим текстовый файл с названием test.txt в папке C:\WebServers. Затем запишем в файл строку "Content". Теперь получим содержимое файла с помощью функции LOAD_FILE():

  ```
  SELECT LOAD_FILE('C:/WebServers/test.txt');
  ```

  Вывод: Content.

4.8.4. Функции для шифрования строк

Функции для необратимого шифрования:

- MD5(<Строка>) — кодирует строку, используя алгоритм MD5. Возвращает 32-разрядное шестнадцатеричное число. Используется для кодирования паролей, т. к. не существует алгоритма для дешифровки. Для сравнения введенного пользователем пароля с сохраненным в базе данных необходимо зашифровать введенный пароль, а затем произвести сравнение:

  ```
  SELECT MD5('Пароль');
  ```

  Вывод: 4a9866c3070171aa5a9faab83e61d887;

- PASSWORD(<Строка>) — используется для шифрования паролей в таблице привилегий MySQL:

  ```
  SELECT PASSWORD('Пароль');
  ```

  Вывод: 6a5b96720c1b5957;

- SHA1(<Строка>) — возвращает 40-разрядное шестнадцатеричное число:

  ```
  SELECT SHA1('Пароль');
  ```

  Вывод: 8affbaf9a316d8b5500236c3daa1ce54a5a0385d;

- SHA(<Строка>) — возвращает 40-разрядное шестнадцатеричное число:

  ```
  SELECT SHA('Пароль');
  ```

  Вывод: 8affbaf9a316d8b5500236c3daa1ce54a5a0385d;

- ENCRYPT(<Строка>, [<Ключ>]) — если параметр <Ключ> не указан, то функция каждый раз будет возвращать разный результат. В операционной системе Windows функция всегда возвращает значение NULL.

Функции для симметричного шифрования:

- AES_ENCRYPT(<Строка>, <Ключ>) — функция принимает строку и секретный ключ. Возвращает зашифрованную строку:

  ```
  SELECT AES_ENCRYPT('Пароль', 'Ключ');
  ```

- AES_DECRYPT(<Зашифрованная строка>, <Ключ>) — служит для расшифровки строк, зашифрованных функцией AES_ENCRYPT():

  ```
  SELECT AES_DECRYPT(AES_ENCRYPT('Пароль', 'Ключ'), 'Ключ');
  ```

  Вывод: Пароль;

- ENCODE(<Строка>, <Ключ>) — функция принимает строку и секретный ключ. Возвращает зашифрованную строку:

  ```
  SELECT ENCODE('Пароль', 'Ключ');
  ```

- DECODE(<Зашифрованная строка>, <Ключ>) — служит для расшифровки строк, зашифрованных функцией ENCODE():

  ```
  SELECT DECODE(ENCODE('Пароль', 'Ключ'), 'Ключ');
  ```

  Вывод: Пароль;

- DES_ENCRYPT(<Строка>, [<Номер ключа>] | [<Ключ>]) — функция принимает строку и секретный ключ (или номер записи в ключевом DES-файле сервера). Возвращает зашифрованную строку. Если в MySQL не включена поддержка SSL, то функции DES_ENCRYPT() и DES_DECRYPT() возвращают NULL;

- DES_DECRYPT(<Зашифрованная строка>, <Ключ>) — служит для расшифровки строк, зашифрованных функцией DES_ENCRYPT().

4.8.5. Информационные функции

Информационные функции:

- VERSION() — возвращает информацию о версии сервера MySQL:

  ```
  SELECT VERSION();
  ```

  Вывод: 5.0.45-community-nt;

- `USER()` — возвращает имя пользователя и имя хоста текущего пользователя:

```
SELECT USER();
```

Вывод: `root@localhost`;

- `CURRENT_USER()` — возвращает имя пользователя и имя хоста текущего пользователя в сессии:

```
SELECT CURRENT_USER();
```

Вывод: `root@localhost`;

- `DATABASE()` — возвращает название текущей базы данных:

```
USE testDB;
SELECT DATABASE();
```

Вывод: `testDB`;

- `CONNECTION_ID()` — возвращает идентификатор соединения:

```
SELECT CONNECTION_ID();
```

Вывод: `1`;

- `DEFAULT(<Имя поля>)` — возвращает значение по умолчанию для указанного поля:

```
USE testDB;
CREATE TABLE new_table (
id int AUTO_INCREMENT,
count int DEFAULT 25,
PRIMARY KEY(id)
) ENGINE=MyISAM;
INSERT INTO new_table VALUES(NULL, 50);
SELECT DEFAULT(count) FROM new_table LIMIT 1;
```

Вывод: `25`;

- `LAST_INSERT_ID()` — возвращает последнее автоматически сгенерированное значение для поля с атрибутом `AUTO_INCREMENT`. Значение возвращается только в том случае, если перед вызовом функции было сгенерировано новое значение:

```
INSERT INTO new_table VALUES(NULL, 80);
SELECT LAST_INSERT_ID();
```

Вывод: `2`.

Вывести последнюю добавленную запись можно следующими способами:

```
SELECT count FROM new_table WHERE id = LAST_INSERT_ID();
```

Вывод: `80`.

```
SELECT count FROM new_table WHERE id IS NULL;
```

Вывод: 80;

- FOUND_ROWS() — возвращает количество строк, которое возвратил бы оператор SELECT без инструкции LIMIT. Чтобы получить значение, необходимо в операторе SELECT указать опцию SQL_CALC_FOUND_ROWS:

```
INSERT INTO new_table VALUES(NULL, 6), (NULL, 70), (NULL, 50);
SELECT COUNT(*) FROM new_table;
```

Вывод: 5.

```
SELECT SQL_CALC_FOUND_ROWS count FROM new_table LIMIT 0, 3;
```

Вывод: три записи.

```
SELECT FOUND_ROWS();
```

Вывод: 5;

- BENCHMARK(<Число повторений>, <SQL-запрос>) — выполняет SQL-запрос заданное количество раз. Функция всегда возвращает значение 0. Используется для определения быстродействия SQL-запроса:

```
SELECT BENCHMARK(1000000,MD5('строка'));
```

Вывод:

```
0
1 row in set (2.89 sec)
```

4.8.6. Прочие функции

Прочие функции:

- IF(<Условие>, <Если Истина>, <Если Ложь>) — функция для логического выбора. Если <Условие> истинно, то возвращается выражение <Если Истина>, в противном случае возвращается выражение <Если Ложь>:

```
SELECT IF(5>6, 'Больше', 'Меньше');
```

Вывод: Меньше;

- CASE() — функция для логического выбора. Имеет две формы записи. Первая форма:

```
CASE <Переменная или выражение>
WHEN <Значение 1> THEN <Выражение 1>
[WHEN <Значение 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END
```

В зависимости от значения переменной (или выражения) выполняется один из блоков WHEN, в котором указано это значение. Если ни одно из значений не описано в блоках WHEN, то выполняется блок ELSE:

```
SELECT CASE 3 + 5 WHEN 8 THEN 'Равно 8'
WHEN 7 THEN 'Равно 7' ELSE 'Не смогли определить' END;
```

Вывод: Равно 8.

Вторая форма:

```
CASE WHEN <Условие 1> THEN <Выражение 1>
[WHEN <Условие 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END

SELECT CASE WHEN 5>6 THEN 'Больше' ELSE 'Меньше' END;
```

Вывод: Меньше;

- IFNULL(<Выражение1>, <Выражение2>) — функция для логического выбора. Если <Выражение1> не равно NULL, то функция возвращает <Выражение1>. В противном случае функция возвращает <Выражение2>:

```
SELECT IFNULL(5, 3);
```

Вывод: 5.

```
SELECT IFNULL(NULL, 3);
```

Вывод: 3;

- NULLIF(<Выражение1>, <Выражение2>) — функция для логического выбора. Если <Выражение1> равно <Выражение2>, возвращается значение NULL, в противном случае функция возвращает <Выражение1>:

```
SELECT NULLIF(5, 5);
```

Вывод: NULL.

```
SELECT NULLIF(5, 3);
```

Вывод: 5;

- INET_ATON(<IP-адрес>) — представляет IP-адрес в виде целого числа:

```
SELECT INET_ATON('127.0.0.1');
```

Вывод: 2130706433;

- INET_NTOA(<IP-адрес в виде числа>) — функция принимает IP-адрес в виде целого числа и возвращает IP-адрес в виде строки, состоящей из четырех чисел, разделенных точками:

```
SELECT INET_NTOA(2130706433);
```

Вывод: 127.0.0.1;

- GET_LOCK(<Имя>, <Время ожидания ответа сервера>) — устанавливает блокировку с указанным именем. Функция возвращает 1 в случае успешной блокировки и 0, если время ожидания ответа сервера превысило величину, указанную в параметре <Время ожидания ответа сервера>. Если произошла ошибка, то функция возвращает NULL.

  ```
  SELECT GET_LOCK('mylock', 5);
  ```

  Вывод: 1.

  Блокировка снимается тремя способами:

  - с помощью функции RELEASE_LOCK();
  - при повторном вызове функции GET_LOCK();
  - при разрыве соединения с сервером;

- IS_FREE_LOCK(<Имя блокировки>) — проверяет, свободна ли блокировка с указанным именем. Функция возвращает 1, если блокировка свободна, и 0, если занята;

  ```
  SELECT IS_FREE_LOCK('mylock');
  ```

  Вывод: 0;

- IS_USED_LOCK(<Имя блокировки>) — функция проверяет, установлена ли блокировка с указанным именем. Если блокировка установлена, то возвращается идентификатор соединения клиента, который установил блокировку. Если блокировка не установлена, то возвращается значение NULL:

  ```
  SELECT IS_USED_LOCK('mylock');
  ```

  Вывод: 1.

  ```
  SELECT CONNECTION_ID();
  ```

  Вывод: 1;

- RELEASE_LOCK(<Имя блокировки>) — функция снимает блокировку с указанным именем. Если блокировка успешно снята, то функция возвращает 1. Если блокировка не может быть снята, то возвращает 0. Если блокировка с указанным именем не существует, то функция возвращает NULL:

  ```
  SELECT RELEASE_LOCK('mylock');
  ```

  Вывод: 1.

  ```
  SELECT IS_USED_LOCK('mylock');
  ```

  Вывод: NULL;

- UUID() — функция возвращает универсальный уникальный идентификатор — 128-разрядное уникальное число в виде строки, состоящее из пяти шестнадцатеричных чисел, разделенных символом дефисом:

```
SELECT UUID();
```

Вывод: `ecbeca8c-d967-102b-8a38-cfef42799994`.

```
SELECT UUID();
```

Вывод: `fa0cea24-d967-102b-8a38-cfef42799994`;

- `GROUP_CONCAT()` — объединяет отдельные значения в одну строку. Функция имеет следующий формат:

```
GROUP_CONCAT([DISTINCT] <Поле1> [, <ПолеN>]
[ORDER BY <Поле> [ASC | DESC]]
[SEPARATOR <Разделитель>]
)
```

В качестве примера создадим таблицу `concat_table` в базе данных testDB:

```
CREATE TABLE concat_table (
   id int NOT NULL auto_increment,
   counter int,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим несколько записей:

```
INSERT INTO concat_table VALUES (NULL, 10);
INSERT INTO concat_table VALUES (NULL, 20);
INSERT INTO concat_table VALUES (NULL, 30);
INSERT INTO concat_table VALUES (NULL, 40);
INSERT INTO concat_table VALUES (NULL, 20);
```

Теперь продемонстрируем возможности функции `GROUP_CONCAT()`. Выведем все значения поля `counter`:

```
SELECT GROUP_CONCAT(counter) FROM concat_table;
```

Вывод: `10,20,30,40,20`.

А теперь выведем только уникальные значения, отсортированные в порядке убывания:

```
SELECT GROUP_CONCAT(DISTINCT counter ORDER BY counter DESC)
FROM concat_table;
```

Вывод: `40,30,20,10`.

И, наконец, выведем все значения поля `counter` больше 10 через точку с запятой:

```
SELECT GROUP_CONCAT(DISTINCT counter
ORDER BY counter ASC SEPARATOR '; ')
FROM concat_table WHERE counter > 10;
```

Вывод: `20; 30; 40`.

4.9. Переменные SQL

Результат текущего запроса можно сохранить в переменной и использовать в последующих запросах в рамках одного сеанса. Присвоить значение переменной можно следующими способами:

- с помощью оператора `SELECT`:

```
SELECT @time := NOW();
```

Вывод: `2008-09-22 00:02:37`.

```
SELECT @time;
```

Вывод: `2008-09-22 00:02:37`;

- с помощью оператора `SET`:

```
SET @time = NOW();
SELECT @time;
```

Вывод: `2008-09-22 00:03:18`.

Объявление переменной начинается с символа `@`, а сохранить значение в переменной позволяет оператор `:=`. Обратите внимание, в случае применения оператора `SET` вместо оператора `:=` можно использовать оператор `=`. Следует также помнить, что имя переменной зависит от регистра символов.

В качестве примера создадим таблицу `var_table` в базе данных testDB:

```
CREATE TABLE var_table (
   id int NOT NULL auto_increment,
   name_tovar varchar(255),
   count int,
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Поле `name_tovar` предназначено для хранения названия товара, а поле `count` служит для обозначения количества товара на складе. Добавим несколько записей:

```
INSERT INTO var_table VALUES (NULL, 'Монитор', 10);
INSERT INTO var_table VALUES (NULL, 'Клавиатура', 20);
INSERT INTO var_table VALUES (NULL, 'Мышь', 30);
INSERT INTO var_table VALUES (NULL, 'Тюнер', 40);
INSERT INTO var_table VALUES (NULL, 'HDD', 20);
```

Сохраним в переменной минимальное количество товара на складе, а затем выведем название товара с минимальным количеством:

```
SELECT @min := MIN(count) FROM var_table;
```

Вывод: `10`.

```
SELECT name_tovar FROM var_table WHERE count = @min;
```

Вывод: `Монитор`.

Если запрос вернет более одного варианта, то в переменной сохранится только последнее значение:

```
SELECT @min := count FROM var_table;
```

Вывод:

```
  10
  20
  30
  40
  20
SELECT @min;
```

Вывод: `20`.

4.10. Временные таблицы

Временные таблицы создаются с помощью оператора `CREATE TEMPORARY TABLE`. Синтаксис оператора ничем не отличается от оператора `CREATE TABLE`. Заполнить временную таблицу можно обычным способом, но чаще всего временные таблицы заполняют с помощью вложенных запросов. В этом случае временная таблица используется для реализации поиска в найденном. Следует помнить, что имя временной таблицы действительно только в течение текущего соединения с сервером. По завершению соединения с сервером временная таблица автоматически удаляется.

В качестве примера создадим таблицу `user_table` в базе данных testDB:

```
CREATE TABLE user_table (
   id int NOT NULL auto_increment,
   name varchar(255),
   PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

В поле `name` мы будем хранить фамилию и имя пользователя. Добавим в таблицу несколько записей:

```
INSERT INTO user_table VALUES (NULL, 'Иванов Сергей');
INSERT INTO user_table VALUES (NULL, 'Иванов Николай');
INSERT INTO user_table VALUES (NULL, 'Иванов Иван');
```

```
INSERT INTO user_table VALUES (NULL, 'Петров Александр');
INSERT INTO user_table VALUES (NULL, 'Иванов Максим');
```

А теперь инсценируем ситуацию поиска в найденном с помощью временных таблиц. Предположим, что первоначальный запрос пользователя выводит клиентов с фамилией Иванов:

```
SELECT id, name FROM user_table WHERE name LIKE '%Иванов%';
```

Сохраним результат запроса во временной таблице, а затем выведем клиентов только с именем Николай:

```
CREATE TEMPORARY TABLE temp
SELECT id, name FROM user_table WHERE name LIKE '%Иванов%';
SELECT name FROM temp WHERE name LIKE '%Николай%';
```

Вывод: `Иванов Николай`.

Обратите внимание, при использовании вложенных запросов не нужно определять структуру временной таблицы. По умолчанию структура временной таблицы будет такой же, как и в результирующей таблице. Посмотреть структуру временной таблицы можно с помощью оператора `DESCRIBE`:

```
CREATE TEMPORARY TABLE temp2
SELECT id, name FROM user_table WHERE name LIKE '%Иванов%';
DESCRIBE temp2;
```

Удалить временную таблицу можно следующими способами:

- с помощью оператора `DROP TABLE`:

  ```
  DROP TABLE <Имя временной таблицы>;
  ```

- по завершению соединения с сервером временная таблица будет удалена автоматически.

4.11. Вложенные запросы

Для изучения вложенных запросов создадим в базе данных testDB следующие таблицы:

- `users_table` — для хранения данных о клиентах:

  ```
  CREATE TABLE users_table (
     id_user int NOT NULL auto_increment,
     name varchar(255),
     PRIMARY KEY (id_user)
  ) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
  ```

- `product_table` — для хранения данных о товарах:

```
CREATE TABLE product_table (
   id_product int NOT NULL auto_increment,
   name_product varchar(255),
   PRIMARY KEY (id_product)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

- `orders_table` — для хранения сведений о покупках:

```
CREATE TABLE orders_table (
   id_orders int NOT NULL auto_increment,
   id_product int,
   id_user int,
   count int,
   PRIMARY KEY (id_orders)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Добавим в таблицы несколько записей:

```
INSERT INTO users_table VALUES (1, 'Иванов');
INSERT INTO users_table VALUES (2, 'Петров');

INSERT INTO product_table VALUES (1, 'Монитор');
INSERT INTO product_table VALUES (2, 'Клавиатура');
INSERT INTO product_table VALUES (3, 'Мышь');

INSERT INTO orders_table VALUES (1, 1, 1, 2);
INSERT INTO orders_table VALUES (2, 3, 2, 5);
INSERT INTO orders_table VALUES (3, 2, 1, 1);
```

4.11.1. Заполнение таблицы с помощью вложенного запроса

При изучении временных таблиц мы уже использовали вложенный запрос для заполнения временной таблицы. Заполнять можно не только временные таблицы, но и обычные таблицы, создаваемые с помощью оператора `CREATE TABLE`. Создадим таблицу `orders_item_table` с помощью вложенного запроса:

```
CREATE TABLE orders_item_table (
   id_orders int NOT NULL auto_increment,
   PRIMARY KEY (id_orders)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
```

```
SELECT orders_table.id_orders AS id_orders,
users_table.name AS user,
product_table.name_product AS name,
orders_table.count AS count
FROM users_table, product_table, orders_table
WHERE orders_table.id_user = users_table.id_user AND
orders_table.id_product = product_table.id_product;
```

Выведем структуру созданной таблицы с помощью SQL-запроса:

```
DESCRIBE orders_item_table\G
```

Вывод:

```
*************************** 1. row ***************************
Field: id_orders
Type: int(11)
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
*************************** 2. row ***************************
Field: user
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
*************************** 3. row ***************************
Field: name
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
*************************** 4. row ***************************
Field: count
Type: int(11)
Null: YES
Key:
Default: NULL
Extra:
```

Как видно из результата, столбцы, не определенные в операторе CREATE TABLE, но имеющиеся в результирующей таблице, добавляются в новую таблицу. Если столбцы определены в операторе CREATE TABLE, но отсутствуют во вложенном запросе, то они получают значение по умолчанию.

Использовать вложенные запросы можно и в операторе INSERT. Создадим таблицу orders_item2_table обычным образом:

```
CREATE TABLE orders_item2_table (
  id_orders int NOT NULL auto_increment,
  user varchar(255),
  name varchar(255),
  count int,
  PRIMARY KEY (id_orders)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим записи с помощью оператора INSERT и вложенного запроса:

```
INSERT IGNORE INTO orders_item2_table
SELECT orders_table.id_orders AS id_orders,
users_table.name AS user,
product_table.name_product AS name,
orders_table.count AS count
FROM users_table, product_table, orders_table
WHERE orders_table.id_user = users_table.id_user AND
orders_table.id_product = product_table.id_product;
```

С помощью ключевых слов IGNORE и REPLACE можно указать, как следует обрабатывать записи с дублированными значениями. При использовании ключевого слова IGNORE повторяющиеся записи отбрасываются, а при использовании REPLACE — новые записи заменяют существующие. Если ни одно из ключевых слов не указано и производится попытка вставить повторяющееся значение, это приведет к ошибке.

4.11.2. Применение вложенных запросов в инструкции *WHERE*

Выведем имя пользователя, сделавшего заказ под номером 2 с помощью вложенного запроса:

```
SELECT name FROM users_table
WHERE id_user = (SELECT id_user FROM orders_table
WHERE id_orders = 2);
```

Вывод: Петров.

В данном примере мы объединили два запроса в один. Первый запрос получает идентификатор пользователя, сделавшего заказ с номером 2, а второй запрос по полученному идентификатору получает имя пользователя. Как видно из примера, вложенный запрос всегда заключается в круглые скобки.

Уровень вложенности запроса может быть более двух. Хотя на практике не используются запросы с уровнем вложенности более трех, т. к. это приводит к увеличению времени выполнения запроса.

Если вложенный запрос возвращает более одного значения, то MySQL генерирует ошибку. Обойти эту проблему можно следующими способами:

- использовать ключевые слова `IN` или `NOT IN`:

```
SELECT name FROM users_table
WHERE id_user IN (SELECT id_user FROM orders_table);
```

Вывод:

```
Иванов
Петров
```

- использовать ключевые слова `ANY` или `SOME`:

```
SELECT name FROM users_table
WHERE id_user > ANY (SELECT id_user FROM orders_table);
```

Вывод: `Петров`;

- использовать ключевое слово `ALL`:

```
SELECT name FROM users_table
WHERE id_user <= ALL (SELECT id_user FROM orders_table);
```

Вывод: `Иванов`.

При использовании ключевого слова `IN` проверяется совпадение с одним из значений, возвращаемых вложенным запросом. При использовании ключевого слова `NOT IN`, наоборот, проверяется отсутствие совпадения со списком значений. Если применяется ключевое слово `ANY`, то проверяемое значение поочередно сравнивается с каждым элементом, и если хотя бы одно сравнение возвращает истинное значение, то результат попадает в итоговую таблицу. В случае использования ключевого слова `ALL` в результирующую таблицу попадут значения, только если все сравнения вернут значение Истина.

С помощью ключевого слова `EXISTS` можно проверить результирующую таблицу на наличие строк. Если вложенный запрос возвращает результат, то ключевое слово `EXISTS` возвращает 1 (Истина). В противном случае возвращается значение 0 (Ложь). Получить противоположные значения позволяет ключевое слово `NOT EXISTS`.

В качестве примера выведем всех клиентов, сделавших хотя бы один заказ. Для наглядности добавим в таблицу `users_table` еще одного клиента:

```
INSERT INTO users_table VALUES (3, 'Сидоров');

SELECT name FROM users_table
WHERE EXISTS (SELECT * FROM orders_table
WHERE orders_table.id_user = users_table.id_user);
```

Вывод:

```
Иванов
Петров
```

А теперь выведем клиентов, не сделавших ни одного заказа:

```
SELECT name FROM users_table
WHERE NOT EXISTS (SELECT * FROM orders_table
WHERE orders_table.id_user = users_table.id_user);
```

Вывод:

```
Сидоров
```

Обратите внимание, внутри вложенного запроса мы указываем таблицу из внешнего запроса:

```
users_table.id_user
```

Такая связь называется *внешней ссылкой*, а сам запрос называется *коррелированным вложенным запросом*.

4.12. Внешние ключи

При эксплуатации реляционной базы данных время от времени необходимо изменить ее структуру. Например, необходимо удалить учетные записи клиентов, которые давно не совершали покупок. Если просто удалить этих клиентов из одной таблицы, то это может привести к нарушению ссылочной целостности базы данных, т. к. кто-нибудь из удаляемых клиентов наверняка совершал ранее покупки, а значит, сведения о покупке заносились в таблицу заказов. По этой причине при удалении клиента необходимо предварительно удалить все записи о совершенных им покупках из таблицы заказов. Сделать это можно с помощью двух SQL-запросов. Первый запрос удаляет записи из таблицы заказов, а второй — удаляет запись о клиенте.

Для таблиц типа `InnoDB` предусмотрена возможность автоматического контроля над ссылочной целостностью базы данных при помощи внешних ключей.

В Денвере тип таблиц `InnoDB` по умолчанию отключен. Чтобы подключить поддержку этого типа таблиц, запускаем Блокнот. В меню **Файл** выбираем пункт **Открыть**. В открывшемся окне в списке **Тип файлов** выбираем пункт

Все файлы. Переходим в папку C:\WebServers\usr\local\mysql5 и открываем файл my.cnf. Находим строку:

```
skip-innodb
```

Заменяем ее на:

```
# skip-innodb
```

Сохраняем файл. Затем перезапускаем Денвер. Для проверки запускаем программу MySQL monitor и в командной строке набираем SQL-команду:

```
SHOW ENGINES;
```

В столбце **Support** напротив типа `InnoDB` должно быть значение "YES". Теперь можно приступать к изучению внешних ключей.

Внешний ключ можно добавить при создании таблицы с помощью оператора `CREATE TABLE`, а при помощи оператора `ALTER TABLE` можно добавить внешний ключ в уже существующую таблицу.

Добавляется внешний ключ с помощью конструкции `FOREIGN KEY`. Конструкция имеет следующий формат:

```
FOREIGN KEY [<Имя ключа>] (<Список полей в текущей таблице>)
REFERENCES <Имя внешней таблицы> (<Список полей во внешней таблице>)
[ON DELETE <Действие>]
[ON UPDATE <Действие>]
```

В параметре `<Действие>` могут быть указаны значения:

- `CASCADE` — удаление или изменение записи, содержащей первичный ключ, приведет к автоматическому удалению или изменению соответствующих записей в таблице-потомке;
- `SET NULL` — при удалении или изменении записи, содержащей первичный ключ, соответствующие записи в таблице-потомке получат значение `NULL`;
- `NO ACTION` — при удалении или изменении записи, содержащей первичный ключ, никаких действий не производится, пока в таблице-потомке существуют ссылающиеся записи;
- `RESTRICT` — нельзя удалить или изменить запись, пока в таблице-потомке существуют ссылающиеся записи.

Для примера создадим в базе данных testDB следующие таблицы:

- `users_foreign` — для хранения данных о клиентах:

```
CREATE TABLE users_foreign (
   id_user int NOT NULL auto_increment,
   name varchar(255),
   PRIMARY KEY (id_user)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

- `product_foreign` — для хранения данных о товарах:

```
CREATE TABLE product_foreign (
   id_product int NOT NULL auto_increment,
   name_product varchar(255),
   PRIMARY KEY (id_product)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

- `orders_foreign` — для хранения сведений о покупках:

```
CREATE TABLE orders_foreign (
   id_orders int NOT NULL auto_increment,
   id_product int,
   id_user int,
   count int,
   PRIMARY KEY (id_orders),
   FOREIGN KEY (id_user) REFERENCES users_foreign (id_user)
   ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (id_product)
   REFERENCES product_foreign (id_product)
   ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

Добавим в таблицы несколько записей:

```
INSERT INTO users_foreign VALUES (1, 'Иванов');
INSERT INTO users_foreign VALUES (2, 'Петров');

INSERT INTO product_foreign VALUES (1, 'Монитор');
INSERT INTO product_foreign VALUES (2, 'Клавиатура');
INSERT INTO product_foreign VALUES (3, 'Мышь');

INSERT INTO orders_foreign VALUES (1, 1, 1, 2);
INSERT INTO orders_foreign VALUES (2, 3, 2, 5);
INSERT INTO orders_foreign VALUES (3, 2, 1, 1);
```

Теперь попробуем удалить господина Иванова из таблицы `users_foreign`:

```
DELETE FROM users_foreign WHERE id_user=1;
SELECT name FROM users_foreign;
```

Вывод: `Петров`.

А теперь посмотрим, сколько заказов осталось в таблице `orders_foreign`:

```
SELECT id_orders FROM orders_foreign;
```

Вывод: `2`.

Как видно из этого примера, удаление господина Иванова привело к автоматическому удалению заказов с его участием за счет применения слова `CASCADE`. Теперь попробуем добавить заказ на уже не существующего господина Иванова:

```
INSERT INTO orders_foreign VALUES (NULL, 1, 1, 2);
```

В итоге получим ошибку:

```
#1452 – Cannot add or update a child row: a foreign key constraint fails
```

Попробуем удалить товар с номером 3 из таблицы `product_foreign`:

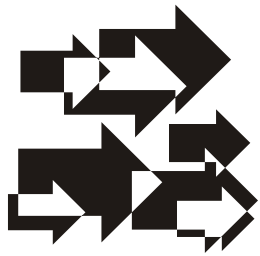
```
DELETE FROM product_foreign WHERE id_product=3;
```

В итоге получим ошибку:

```
#1451 – Cannot delete or update a parent row: a foreign key constraint fails
```

Иными словами, пока мы не удалим заказ с номером 2 из таблицы `orders_foreign`, мы не сможем удалить товар с номером 3 из таблицы `product_foreign`. Это достигается за счет применения слова `RESTRICT`.

ГЛАВА 5



Сплошная практика

5.1. Структура будущего сайта

В качестве примера создадим полностью динамический сайт с использованием Perl и MySQL, например, каталог ресурсов Интернета. Сайт будет управляться из одного файла и состоять из трех разделов — общедоступного, раздела для зарегистрированных пользователей и раздела для администратора сайта.

Для тестирования сайта создадим виртуальный хост site.ru. Для этого создаем папку site.ru в каталоге C:\WebServers\home. Внутри новой папки создаем каталоги:

- www — для файлов в формате HTML и картинок;
- cgi-bin — для скриптов, написанных на языке Perl.

> ***Обратите внимание***
> Чтобы изменения вступили в силу, необходимо перезапустить Денвер.

В каталоге /www/ создадим файлы:

- .htaccess — для обработки ошибок;
- index.shtml — центральная страница сайта.

В общедоступном разделе (каталог /cgi-bin/) создадим следующие файлы:

- index.pl — центральная страница сайта, на которой выводится рубрикатор каталога;
- catalog.pl — вывод сайтов по рубрике с разбиением на страницы;
- search.pl — для поиска по каталогу с разбиением на страницы;
- contact.pl — страница с формой обратной связи;

- gbook.pl — гостевая книга с разбиением на страницы;
- err401.pl — файл с сообщением об ошибке 401 (не авторизован);
- err403.pl — файл с сообщением об ошибке 403 (нет доступа);
- err404.pl — файл с сообщением об ошибке 404 (ресурс не найден);
- err401.txt — для регистрации ошибки 401;
- err403.txt — для регистрации ошибки 403;
- err404.txt — для регистрации ошибки 404;
- errlog.txt — для регистрации всех ошибок в скриптах;
- .htaccess — для управления доступом и обработки ошибок.

В разделе для зарегистрированных пользователей (каталог /cgi-bin/user/) создадим файлы:

- index.pl — страница с формой входа в систему, с формой регистрации нового пользователя и формой восстановления пароля;
- add.pl — для регистрации нового сайта;
- exit.pl — для выхода из системы.

Кроме того, для хранения временных файлов сессии создадим в каталоге /cgi-bin/user/ папку tmp.

В разделе для администратора сайта (каталог /cgi-bin/admin/) создадим файлы:

- index.pl — вывод статистики и сообщений об ошибках;
- rubr.pl — для добавления новой рубрики, удаления или переименования существующей;
- moder.pl — для вывода сайтов, находящихся на модерации, и результатов поиска по URL-адресу с возможностью одобрения или удаления выделенных сайтов;
- catalog.pl — для редактирования описания сайта;
- gbook.pl — вывод содержимого гостевой книги с выделением новых сообщений, а также возможностью одобрения или удаления выделенных сообщений.

Дополнительно в каталоге /cgi-bin/config/ создадим файл конфигурации Allscript.pm в виде модуля, который будет содержать все функции. С помощью редактирования этого файла можно изменить весь дизайн сайта. Местоположение модуля будем указывать во всех файлах с помощью прагмы `lib`:

```
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
```

```
# Подключаем файл конфигурации
use Allscript;
```

В модуле опишем постоянные величины, которые будем использовать при создании страниц (листинг 5.1). Для хранения этих констант используем глобальный ассоциативный массив `%DATA`.

Листинг 5.1. Описание констант

```
our %DATA;
# URL-адрес сайта (например, http://www.site.ru/)
$DATA{'URL_SITE'} = "http://site.ru/";
$DATA{'WIDTH_TABLE_1'} = "760";  # Ширина таблиц 1-го уровня
$DATA{'WIDTH_TABLE_2'} = "600";  # Ширина таблиц 2-го уровня
$DATA{'WIDTH_TABLE_3'} = "100%";  # Ширина таблиц 3-го уровня

# Данные для подключения к базе данных
$DATA{'HOST'} = "localhost"; # Сервер
$DATA{'LOGIN'} = "root"; # Логин
$DATA{'PASSW'} = ""; # Пароль
$DATA{'DB'} = "CatalogDB"; # База данных

# Данные для писем (например, support <unicross\@mail.ru>)
$DATA{'MAIL_POST'} = "support <unicross\@mail.ru>";
# Для формы обратной связи (например, unicross\@mail.ru)
$DATA{'MAIL_ADRES'} = "unicross\@mail.ru";

# Время жизни сессии
$DATA{'TIME_SESS'} = "+1h";
# Местоположение файлов сессии
$DATA{'TMP_PATH'} = "C:/WebServers/home/site.ru/cgi-bin/user/tmp/";

# Количество сообщений в гостевой книге
$DATA{'count_pos_page_gbook'} = 10;
# Количество сайтов на странице
$DATA{'count_pos_page_site'} = 10;
```

Как вы уже знаете, скрипты на языке Perl должны располагаться в специальном каталоге (обычно cgi-bin). Это означает, что мы не можем разместить скрипты в каталоге www, а значит, нет возможности создать центральную страницу сайта на языке Perl. Конечно, можно просто автоматически перена-

правлять посетителей на файл /cgi-bin/index.pl, разместив в каталоге www файл index.html со следующим содержимым:

```
<HTML>
<HEAD>
<META http-equiv="refresh" content="0; /cgi-bin/index.pl">
</HEAD>
<BODY></BODY>
</HTML>
```

Вместо файла index.html можно разместить файл index.php. Содержимое файла выглядит следующим образом:

```
<?php
Header("Location: /cgi-bin/index.pl");
exit();
?>
```

Все эти методы заставляют Web-браузер повторно сделать запрос. Web-сервер Apache предоставляет более удобный способ решения этой проблемы — технологию *SSI* (Server Side Includes, включения на стороне сервера). Технология SSI представляет собой набор простых команд в тексте HTML-документа. Эти команды выполняются на сервере до передачи ответа Web-браузеру. Именно этой технологией мы и воспользуемся для создания центральной страницы сайта.

Обычно в настройках Web-сервера с технологией SSI связываются файлы с расширением SHTML. В Денвере по умолчанию команды SSI обрабатываются в файлах с расширениями HTML и SHTML. Это достигается следующим кодом в файле конфигурации Web-сервера Apache:

```
AddOutputFilter INCLUDES .shtml .html
```

Для включения результатов выполнения cgi-программы используются следующие команды:

```
<!--#exec cgi="/cgi-bin/index.pl"-->
<!--#include virtual="/cgi-bin/index.pl"-->
```

На практике чаще всего используется команда `include`. Именно этой командой мы и воспользуемся. Для вывода центральной страницы в каталоге www создадим файл index.shtml со следующим кодом:

```
<!--#include virtual="/cgi-bin/index.pl"-->
```

Теперь при запросе **http://site.ru/** мы получим результат выполнения программы /cgi-bin/index.pl.

Чтобы перехватывать ошибки, необходимо в каталоге www создать файл .htaccess.

> ***Обратите внимание***
>
> У файла .htaccess отсутствует название. Только расширение htaccess. Создание файла с названием .htaccess.txt является самой распространенной ошибкой.

Действие файла .htaccess распространяется на все папки, расположенные выше по иерархии, и не распространяется на папки, расположенные ниже. Содержимое файла должно выглядеть так:

```
ErrorDocument 401 /cgi-bin/err401.pl
ErrorDocument 403 /cgi-bin/err403.pl
ErrorDocument 404 /cgi-bin/err404.pl
DirectoryIndex index.shtml index.html
```

Директива `ErrorDocument` позволяет связать код ошибки с определенным обработчиком или просто файлом. Обратите внимание, что дизайн указанного файла должен соответствовать дизайну всего сайта, а все ссылки в этом файле должны иметь абсолютный URL-адрес. Это касается и адресов картинок. В противном случае получите множество сообщений об ошибке 404 (файл не найден).

С помощью директивы `DirectoryIndex` можно задать название файла, который будет выдаваться сервером по умолчанию. Возможно указание нескольких имен файлов через пробел. В этом случае сервер отобразит первый существующий файл из списка. Если ни один файл не найден, то сервер выдаст ошибку 403 (нет доступа).

Так как каталог cgi-bin расположен вне каталога www, то необходимо создать дополнительно файл .htaccess в каталоге cgi-bin и продублировать директивы обработки ошибок. Содержимое файла должно выглядеть следующим образом:

```
ErrorDocument 401 /cgi-bin/err401.pl
ErrorDocument 403 /cgi-bin/err403.pl
ErrorDocument 404 /cgi-bin/err404.pl
<Files *.txt>
   Deny from all
</Files>
<Files *.pm>
   Deny from all
</Files>
```

```
<Files cgisess*>
   Deny from all
</Files>
```

Назначение директивы `ErrorDocument` вы уже знаете. Кроме этой директивы мы указываем три раздела `Files`, внутри которых для директивы `Deny` устанавливаем значение `all`. Раздел `Files` указывает, что директивы применимы только к определенным файлам. Символ `*` соответствует любой последовательности символов, а символ `?` — любому одиночному символу. Директива `Deny` позволяет запретить доступ. При указании значения `all` доступ к файлу закрыт для всех пользователей. Таким образом, мы закрываем доступ к файлам с расширением TXT (логи ошибок) и PM (конфигурационный файл), а также к файлам, название которых начинается с "cgisess" (временные файлы сессий). Теперь при попытке получить запрещенный файл сервер выдаст сообщение об ошибке 403 (нет доступа).

5.2. Создание шаблона сайта

Создание любого сайта начинается с разработки дизайна, т. к. все страницы должны быть одинаково оформлены. Шаблон сайта будет состоять из трех частей:

- верхний колонтитул — в этой части мы разместим логотип сайта, рекламный баннер, панель навигации и поисковую форму, а также таблицу стилей и функции на JavaScript;
- основная часть страницы — содержимое этого раздела будет меняться в зависимости от назначения страницы;
- нижний колонтитул — содержимое этого раздела будет разным для пользователя и администратора. Для пользователей раздел будет содержать только панель навигации, а для администратора — поисковую форму и специальную панель навигации. В дальнейшем в этом разделе можно разместить счетчики и логотипы различных каталогов.

Все части сайта опишем в виде отдельных функций, которые разместим в модуле Allscript.pm. Этот модуль мы будем подключать во всех файлах с помощью оператора `use`. Для удобства вызова функций экспортируем часто используемые идентификаторы, указав их в массиве `@EXPORT`. Идентификаторы, специфические для конкретных скриптов, укажем в массиве `@EXPORT_OK`. Кроме того, внутри модуля определим два тега "user" и "admin" в ассоциативном массиве `%EXPORT_TAGS`. С помощью тега "user" опишем функции для пользователей, а с помощью тега "admin" — для администратора. Для добавления идентификаторов, перечисленных в тегах, в массив `@EXPORT_OK` используем метод `export_ok_tags()` из модуля `Exporter`:

```
# Экспортируем идентификаторы из модуля
use Exporter;
our @ISA = qw( Exporter );
our @EXPORT = qw( %DATA &f_table_2_start &f_table_2_end
                              &f_table_page_start &f_table_page_end );
our @EXPORT_OK = qw( &f_passw_generator &f_table_site &f_table_gbook
                                    &f_date_gm );
our %EXPORT_TAGS = (
   "user" => [qw( &f_header_all &f_footer_user )],
   "admin" => [qw( &f_header_all &f_footer_admin &f_menu_admin )]
);
Exporter::export_ok_tags('user', 'admin');
```

Теперь мы можем указать эти идентификаторы в списке импорта:

```
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_table_site );
```

Тег `:DEFAULT` позволяет импортировать все идентификаторы из массива `@EXPORT`.

Шаблон страниц сайта для пользователей будет выглядеть, как показано в листинге 5.2.

Листинг 5.2. Шаблон страниц сайта для пользователей

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user );
print "Content-type: text/html; charset=windows-1251\n\n";
# Заголовок
my $title = "";
```

```
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);

print "<DIV align=center>Основное содержание страницы</DIV>";

# Выводим нижний колонтитул
&f_footer_user();
```

Для администратора сайта шаблон будет немного другим (листинг 5.3).

Листинг 5.3. Шаблон страниц сайта для администратора

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin );
print "Content-type: text/html; charset=windows-1251\n\n";
# Заголовок
my $title = "";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);

print "<DIV align=center>Основное содержание страницы</DIV>";

# Выводим нижний колонтитул
&f_footer_admin();
```

Заголовок страницы, описание содержимого страницы и ключевые слова должны отличаться. Для этого мы передаем эти параметры в функцию `f_header_all()`, создающую верхний колонтитул.

Мы не выводим сообщения об ошибках в окно Web-браузера, т. к. чем меньше пользователь получает сведений об ошибках, тем лучше. Ведь злоумышленник может использовать эту информацию против нас. Однако администратор должен знать об ошибках. Для вывода сообщений в указанный файл (а не в стандартный журнал ошибок) мы передаем функции `carpout()` ссылку на дескриптор ранее открытого файла. Чтобы получить сообщения об ошибках на этапе компиляции, функция размещена в блоке `BEGIN`.

5.3. Создание верхнего колонтитула

Верхний колонтитул выводится с помощью функции `f_header_all()`. Функция имеет следующий формат:

```
f_header_all(<1>, <2>, <3>, <4>, <5>)
```

- `<1>` — заголовок страницы (тег `<TITLE>`);
- `<2>` — описание страницы (тег `<META>` параметр `description`);
- `<3>` — ключевые слова для поисковых машин (тег `<META>` параметр `keywords`);
- `<4>` — если параметр равен 1, то страница разрешена для индексации поисковыми роботами (значение по умолчанию). В страницу будет вставлена строка:

  ```
  <META name="robots" content="index, follow">
  ```

  Если любое другое значение, то индексация запрещена. В страницу будет вставлена строка:

  ```
  <META name="robots" content="noindex, nofollow">
  ```

- `<5>` — поисковая фраза, которая отобразится в строке поисковой формы. По умолчанию пустая строка.

На самом деле `f_header_all()` лишь промежуточная функция, которая запускает на выполнение несколько других функций (листинг 5.4).

Листинг 5.4. Функция `f_header_all()`

```
sub f_header_all {
   my($t, $d, $k, $r, $s) = @_;
   $r = 1 unless (defined($r));
   $s = "" unless (defined($s));
```

```
    # Заголовок для всех страниц
    f_header($t, $d, $k, $r); # Выводим верхний колонтитул
    &f_logo(); # Выводим логотип и баннер
    &f_menu(); # Выводим панель навигации
    &f_table(); # Выводим таблицу-разделитель
    &f_search($s); # Выводим форму поиска
    &f_table(); # Выводим таблицу-разделитель
    &f_table_center_start(); # Начало таблицы для основного содержания
}
```

Рассмотрим каждую функцию по отдельности.

5.3.1. Вывод верхнего колонтитула

Для вывода верхнего колонтитула используется функция `f_header()` (листинг 5.5).

Листинг 5.5. Функция `f_header()`

```
sub f_header {
   my($t, $d, $k, $r) = @_;
   $r = 1 unless (defined($r));
   # Верхний колонтитул
   print "<HTML>\n";
   print "<HEAD>\n";
   print "<TITLE>$t</TITLE>\n";
   print "<META name=\"description\" content=\"$d\">\n";
   print "<META name=\"keywords\" content=\"$k\">\n";
   print "<META http-equiv=\"content-type\" ";
   print "content=\"text/html; charset=windows-1251\">\n";
   if ($r == 1) {
      print "<META name=\"robots\" content=\"index, follow\">\n";
   }
   else {
      print "<META name=\"robots\" content=\"noindex, nofollow\">\n";
   }
   &f_style(); # Выводим таблицу стилей
   print "<SCRIPT language=\"javascript\">\n";
   print "<!--\n";
   print " window.status=\"$t\";\n";
```

```
   print " function f_add_favor() {\n";
   print "    external.addFavorite(\"";
   print $DATA{'URL_SITE'} . "\", \"$t\");\n";
   print "    return false;\n";
   print " }\n";
   print " function f_HomePage(m_obj) {\n";
   print "    m_obj.style.behavior=\"url(#default#homepage)\";\n";
   print "    m_obj.setHomePage(\"" . $DATA{'URL_SITE'} . "\");\n";
   print "    return false;\n";
   print " }\n";
   print "//-->\n";
   print "</SCRIPT>\n";
   print "</HEAD>\n";
   print "<BODY>\n\n";
}
```

Итак, функция `f_header()` выводит полностью раздел `HEAD` HTML-документа, включая заголовок, описание страницы и ключевые слова для поиска, кодировку страницы и информацию о возможности индексации. Кроме того, с помощью функций JavaScript заголовок выводится в статусной строке Web-браузера, функция `f_add_favor()` позволяет добавить сайт в Избранное, а функция `f_HomePage()` позволяет сделать главную страницу сайта стартовой (первой страницей, которую увидит пользователь при запуске Web-браузера).

Внутри функции `f_header()` используется константа `$DATA{'URL_SITE'}`, которая содержит URL-адрес сайта, а все кавычки внутри оператора `print` мы экранируем с помощью защитных слэшей, т. к. иначе интерпретатор выведет сообщение об ошибке.

5.3.2. Создание таблицы стилей

Внутри функции `f_header()` вызывается функция `f_style()`, которая выводит таблицу стилей (листинг 5.6).

Листинг 5.6. Функция `f_style()`

```
sub f_style {
   print <<STYLE_COD;

<!-- Таблица стилей. Начало -->
<STYLE type="text/css">
   A:link { text-decoration: none; font-weight: bold; color: #000000 }
```

```
A:visited { text-decoration: none; font-weight: bold; color: #000000 }
A:hover { text-decoration: underline; font-weight: bold;
color: #000000 }
BODY { font-family: "Verdana", "Tahoma", sans-serif; font-size: 11px }
BODY { margin-top: 0; background-color: #FFFFFF }
BODY {
   scrollbar-arrow-color: #FFFFFF;
   scrollbar-base-color: #43568E;
   scrollbar-track-color: #E8E8E8;
   scrollbar-darkshadow-color: #FFFFFF
}
TABLE {
   font-family: "Verdana", "Tahoma", sans-serif;
   font-size: 12px;
   background-color: #FFFFFF
}
H1 {
   font-family: "Tahoma", "Verdana", sans-serif;
   font-size: 16px;
   font-weight: bold;
   text-align: center
}
.err {
   font-size:12px;
   font-weight: bold;
   text-align: center;
   color: #FF0000
}
.ok {
   font-size:12px;
   font-weight: bold;
   text-align: center;
   color: #008000
}
.bold { font-weight: bold }
.color_table { background-color: #E8EAF1 }
.logo-table { text-align: center; vertical-align: middle }
```

```
.menu-table {
   text-align: center;
   vertical-align: middle;
   background-color: #43568E
}
A.menu:link {
   text-decoration: none;
   font-weight: bold;
   color: #FFFFFF;
   font-size: 11px;
   font-family: "Verdana", "Tahoma", sans-serif
}
A.menu:visited {
   text-decoration: none;
   font-weight: bold;
   color: #FFFFFF;
   font-size: 11px;
   font-family: "Verdana", "Tahoma", sans-serif
}
A.menu:hover {
   text-decoration: underline;
   font-weight: bold;
   color: #FFFFFF;
   font-size: 11px;
   font-family: "Verdana", "Tahoma", sans-serif
}
.search-table {
   text-align: center;
   vertical-align: middle;
   background-color: #FFFFFF
}
.txt_frm {
   background-color: #FFFFFF;
   font-size: 8pt;
   color: #000000;
   font-weight: bold;
   border-bottom: 1px solid;
   border-right: 1px solid;
```

```
    border-left: 1px solid;
    border-top: 1px solid;
    font-family: "Verdana", "Tahoma", sans-serif
}
.textarea_frm {
    background-color: #FFFFFF;
    font-weight: bold;
    font-size: 8pt;
    color: #000000;
    border-bottom: 1px solid;
    border-right: 1px solid;
    border-left: 1px solid;
    border-top: 1px solid;
    font-family: "Verdana", "Tahoma", sans-serif
}
.select_frm {
    margin-top: 3px;
    background-color: #FFFFFF;
    font-family: "Verdana", "Tahoma";
    font-size: 8pt;
    font-weight: bold
}
.text {
    font-family: "Verdana", "Tahoma", sans-serif;
    font-size: 12px;
    color: #000000;
    vertical-align: top;
    border-right-width: 1px;
    border-right-style: solid;
    border-right-color: silver;
    border-bottom-width: 1px;
    border-bottom-style: solid;
    border-bottom-color: silver;
    border-left-width: 1px;
    border-left-style: solid;
    border-left-color: silver;
    border-top-width: 1px;
```

```
      border-top-style: solid;
      border-top-color: silver
   }
   .search_frm { margin-top: 0; margin-bottom: 0 }
</STYLE>
<!-- Таблица стилей. Конец -->

STYLE_COD
}
```

Итак, с помощью псевдостилей гиперссылок мы задаем внешний вид всех ссылок в документе. Для "тела" документа задаем размер и тип шрифта, цвет фона и внешний вид полосы прокрутки. Определяем стиль по умолчанию для таблиц и заголовков первого уровня. Для сообщений об ошибках создаем стиль `.err`, а для сообщений об удачной операции — стиль `.ok`. Кроме того, для выделения фрагментов текста определим стиль `.bold` и будем использовать его вместо тега `<B>`.

Для таблицы, используемой при форматировании панели навигации, определяем стиль `.menu-table`, а внешний вид ссылок задаем стилем `.menu`.

С помощью стиля `.txt_frm` описываем текстовое поле, а с помощью `.textarea_frm` — поле для ввода многострочного текста. Для списков определен стиль `.select_frm`.

Для ячейки таблицы, содержащей логотип и баннер, предназначен стиль `.logo-table`. Ячейку, в которой находится поисковая форма, описывает стиль `.search-table`, а для поисковой формы предназначен стиль `.search_frm`. И наконец, ячейка таблицы, в которой находится основное содержимое страницы, описывается с помощью стиля `.text`.

5.3.3. Вывод логотипа и рекламного баннера

Для размещения логотипа и баннера на странице используем таблицу высотой 70 пикселов. Ширина таблицы задается константой `$DATA{'WIDTH_TABLE_1'}`. Так как код рекламного баннера содержит много кавычек, используем следующий способ:

```
print <<BANNERCODE;
 
BANNERCODE
```

Весь код баннера должен быть расположен между метками `BANNERCODE`. А т. к. пока у нас нет баннера, то вместо кода запишем неразрывный пробел. Для

размещения логотипа необходимо создать еще одну ячейку справа или слева от баннера. Для вывода используется функция `f_logo()` (листинг 5.7).

Листинг 5.7. Функция `f_logo()`

```
sub f_logo {
   print "<!-- Логотип и баннер. Начало -->\n";
   print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_1'};
   print "\" align=\"center\" ";
   print "border=\"0\" cellspacing=\"0\" cellpadding=\"0\">\n";
   print "<TR><TD class=\"logo-table\" height=\"70\">\n";
   print <<BANNERCODE;
 
BANNERCODE
   print "</TD></TR></TABLE>\n";
   print "<!-- Логотип и баннер. Конец -->\n\n";
}
```

5.3.4. Вывод панели навигации

Панель навигации выводится с помощью функции `f_menu()` (листинг 5.8). Внутри функции мы используем константу `$DATA{'URL_SITE'}`.

Листинг 5.8. Функция `f_menu()`

```
sub f_menu {
   print "<!-- Панель навигации. Начало -->\n";
   print "<TABLE align=\"center\" width=\"";
   print $DATA{'WIDTH_TABLE_1'} . "\" height=\"21\" ";
   print "border=\"0\" cellpadding=\"0\" cellspacing=\"0\">\n";
   print "<TR class=\"menu-table\"><TD>\n";
   print "<A href=\"" . $DATA{'URL_SITE'};
   print "\" class=\"menu\">На главную</A>\n";
   print "</TD><TD>\n";
   print "<A href=\"" . $DATA{'URL_SITE'};
   print "cgi-bin/user/add.pl\" class=\"menu\">";
   print "Добавить сайт</A>\n";
   print "</TD><TD>\n";
   print "<A href=\"" . $DATA{'URL_SITE'};
```

```
    print "cgi-bin/gbook.pl\" class=\"menu\">";
    print "Гостевая книга</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'};
    print "cgi-bin/contact.pl\" class=\"menu\">";
    print "Обратная связь</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'};
    print "\" onclick=\"return f_add_favor();\" class=\"menu\">";
    print "Добавить в Избранное</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'};
    print "\" onclick=\"return f_HomePage(this);\" class=\"menu\">";
    print "Сделать стартовой</A>\n";
    print "</TD></TR>\n";
    print "</TABLE>\n";
    print "<!-- Панель навигации. Конец -->\n\n";
}
```

5.3.5. Вывод таблицы-разделителя

Функция `f_table()` демонстрирует возможность применения таблицы для точного задания промежутка между элементами страницы (листинг 5.9).

Листинг 5.9. Функция `f_table()`

```
sub f_table {
    print "<!-- Таблица-разделитель. Начало -->\n";
    print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_1'};
    print "\" align=\"center\" ";
    print "border=\"0\" cellspacing=\"0\" cellpadding=\"0\">\n";
    print "<TR><TD height=\"5\"> </TD></TR></TABLE>\n";
    print "<!-- Таблица-разделитель. Конец -->\n\n";
}
```

5.3.6. Вывод поисковой формы

Поисковая форма является неотъемлемой частью любой страницы. Выводится форма с помощью функции `f_search()` (листинг 5.10).

Листинг 5.10. Функция `f_search()`

```
sub f_search {
   my $txt = shift();
   $txt = "" unless (defined($txt));
   print "<!-- Поисковая форма. Начало -->\n";
   print <<SCRIPTTEXT;
<SCRIPT language="javascript">
<!--
function f_submit_src() {
   var m_f = document.frm_src;
   if (m_f.search.value=="") {
      window.alert("Поле не заполнено!");
      m_f.search.focus();
      return false;
   }
   if (m_f.search.value.length<3) {
      window.alert("В поле допустимо не менее 3 символов");
      m_f.search.focus();
      return false;
   }
   if (m_f.search.value.length>50) {
      window.alert("В поле допустимо не более 50 символов");
      m_f.search.focus();
      return false;
   }
return true;
}
//-->
</SCRIPT>

SCRIPTTEXT
   print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_1'};
   print "\"  align=\"center\" ";
   print "border=\"0\" cellspacing=\"0\" cellpadding=\"0\">\n";
   print "<TR><TD class=\"search-table\">\n";
   print "<FORM action=\"" . $DATA{'URL_SITE'};
   print "cgi-bin/search.pl\" name=\"frm_src\" ";
```

```
    print "id=\"frm_src\" class=\"search_frm\" ";
    print "onsubmit=\"return f_submit_src();\">\n";
    print "<SPAN class=\"bold\">Поиск по каталогу: </SPAN>\n";
    print "<INPUT type=\"text\" name=\"search\" ";
    print "value=\"$txt\" size=\"70\"> \n";
    print "<INPUT type=\"submit\" value=\"Найти\">\n";
    print "</FORM>\n";
    print "</TD></TR></TABLE>\n";
    print "<!-- Поисковая форма. Конец -->\n\n";
}
```

При нажатии кнопки **Найти** проверяется значение, введенное в строку поиска, при помощи функции `f_submit_src()`. Если поле не заполнено, а также если длина меньше 3 или больше 50 символов, то отправка данных формы прерывается и пользователю выводится соответствующее сообщение.

5.4. Создание элементов основной части страницы

Вся основная часть страницы будет находиться в ячейке таблицы. Эту таблицу мы будем выводить по частям. Начало таблицы входит в состав верхнего колонтитула (листинг 5.11), а конец таблицы в состав нижнего колонтитула (листинг 5.12).

Листинг 5.11. Функция `f_table_center_start()`

```
sub f_table_center_start {
    print "<!-- Таблица для основного содержания страницы. Начало -->\n";
    print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_1'};
    print "\" height=\"300\" align=\"center\" border=\"0\" ";
    print "cellspacing=\"0\" cellpadding=\"10\">\n";
    print "<TR><TD class=\"text\">\n\n";
}
```

Листинг 5.12. Функция `f_table_center_end()`

```
sub f_table_center_end {
    print "<!-- Таблица для основного содержания страницы. Конец -->\n";
    print "</TD></TR></TABLE>\n\n";
}
```

Кроме того, внутри ячейки основной таблицы мы создадим еще одну таблицу, которая в будущем может быть использована для вывода баннеров 120×240 или 120×600, а также для вывода новостей сайта или другой информации. Таблицу будем выводить также по частям. Начало таблицы посредством функции `f_table_2_start()` (листинг 5.13), а конец таблицы с помощью функции `f_table_2_end()` (листинг 5.14).

Листинг 5.13. Функция `f_table_2_start()`

```
sub f_table_2_start {
   print "<!-- Таблица второго уровня. Начало -->\n";
   print "<TABLE align=\"center\" width=\"" . $DATA{'WIDTH_TABLE_2'};
   print "\" bgcolor=\"#FFFFFF\" ";
   print "border=\"0\" cellpadding=\"2\" cellspacing=\"0\">\n";
   print "<TR><TD valign=\"top\">\n";
}
```

Листинг 5.14. Функция `f_table_2_end()`

```
sub f_table_2_end {
   print "<!-- Таблица второго уровня. Конец -->\n";
   print "</TD></TR></TABLE>\n\n";
}
```

Для вывода описания конкретного сайта создадим функцию `f_table_site()` (листинг 5.15). Функция имеет следующий формат:

```
f_table_site(<URL-адрес>, <Название сайта>, <Описание сайта>)
```

Листинг 5.15. Функция `f_table_site()`

```
sub f_table_site {
   # Описание ресурса
   my($url_site, $title_site, $deskr_site) = @_;
   print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_3'};
   print "\" align=\"center\" ";
   print "border=\"0\" cellspacing=\"0\">\n";
   print "<TR><TD class=\"color_table\">\n";
   print "<A href=\"$url_site\" ";
   print "target=\"_blank\">$title_site</A>\n</TD></TR>\n";
```

```
    print "<TR><TD>\n";
    print "$deskr_site\n</TD></TR></TABLE><BR>\n\n";
}
```

Вывод сообщений в гостевой книге оформим с помощью функции `f_table_gbook()` (листинг 5.16). Функция имеет следующий формат:

```
f_table_gbook(<Дата сообщения>, <Автор>, <Сообщение>);
```

Листинг 5.16. Функция `f_table_gbook()`

```
sub f_table_gbook {
    # Сообщение в гостевой книге
    my($msg_date, $author, $msg) = @_;
    print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_3'};
    print "\" align=\"center\" border=\"0\" cellspacing=\"0\">\n";
    print "<TR><TD class=\"color_table\">\n";
    print "<SPAN class=\"bold\">$msg_date $author</SPAN></TD></TR>\n";
    print "<TR><TD>\n";
    print "$msg\n</TD></TR></TABLE><BR>\n\n";
}
```

Для вывода количества страниц создадим две функции: `f_table_page_start()` (листинг 5.17) и `f_table_page_end()` (листинг 5.18). Первая функция содержит начало таблицы, а вторая — конец.

Листинг 5.17. Функция `f_table_page_start()`

```
sub f_table_page_start {
    # Таблица для количества страниц. Начало
    print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_3'};
    print "\" align=\"center\" border=\"0\" cellspacing=\"0\">\n";
    print "<TR><TD>\n";
}
```

Листинг 5.18. Функция `f_table_page_end()`

```
sub f_table_page_end {
    # Таблица для количества страниц. Конец
    print "</TD></TR></TABLE><BR>\n\n";
}
```

5.5. Создание нижнего колонтитула

Нижний колонтитул будет разным для пользователя и администратора. Для пользователей нижний колонтитул будет содержать только панель навигации, а для администратора — поисковую форму и специальную панель навигации.

5.5.1. Вывод нижнего колонтитула для пользователей

Для пользователей нижний колонтитул выводится с помощью функции `f_footer_user()` (листинг 5.19).

Листинг 5.19. Функция `f_footer_user()`

```
sub f_footer_user {
   # Выводим нижний колонтитул для пользователей
   &f_table_center_end();
   # Выводим конец таблицы для основного содержания
   &f_table(); # Выводим таблицу-разделитель
   &f_menu(); # Выводим панель навигации
   &f_footer(); # Выводим нижний колонтитул
}
```

Практически все функции мы уже рассматривали, за исключением одной — `f_footer()` (листинг 5.20).

Листинг 5.20. Функция `f_footer()`

```
sub f_footer {
   print "\n<!-- Нижний колонтитул -->\n";
   print "</BODY></HTML>\n";
}
```

В дальнейшем в этой функции можно разместить счетчики и логотипы различных каталогов.

5.5.2. Вывод нижнего колонтитула для администратора

Для администратора нижний колонтитул выводится с помощью функции `f_footer_admin()` (листинг 5.21).

Листинг 5.21. Функция `f_footer_admin()`

```
sub f_footer_admin {
   # Выводим нижний колонтитул для администратора
   &f_table_center_end();
   # Выводим конец таблицы для основного содержания
   &f_table(); # Выводим таблицу-разделитель
   &f_search_admin(); # Поисковая форма для администратора
   &f_table(); # Выводим таблицу-разделитель
   &f_menu_admin(); # Выводим панель навигации
   &f_footer (); # Выводим нижний колонтитул
}
```

Для администратора нижний колонтитул содержит две новые функции — `f_search_admin()` и `f_menu_admin()`. Функция `f_search_admin()` выводит поисковую форму для администратора (листинг 5.22).

Листинг 5.22. Функция `f_search_admin()`

```
sub f_search_admin {
   print "<!-- Поисковая форма для администратора. Начало -->\n";
   print "<TABLE width=\"" . $DATA{'WIDTH_TABLE_1'};
   print "\" height=\"25\" align=\"center\" border=\"0\" ";
   print "cellspacing=\"0\" cellpadding=\"0\">\n";
   print "<TR><TD class=\"search-table\"><FORM action=\"moder.pl\" ";
   print "class=\"search_frm\">\n";
   print "<SPAN class=\"bold\">Поиск сайта по URL: </SPAN>\n";
   print "<INPUT type=\"text\" name=\"search\" size=\"70\"> \n";
   print "<INPUT type=\"submit\" value=\"Найти\">\n";
   print "</FORM></TD></TR></TABLE>\n";
   print "<!-- Поисковая форма для администратора. Конец -->\n\n";
}
```

Функция `f_menu_admin()` выводит панель навигации для администратора (листинг 5.23).

Листинг 5.23. Функция `f_menu_admin()`

```
sub f_menu_admin {
   print "<!-- Панель навигации для администратора. Начало -->\n";
   print "<TABLE align=\"center\" width=\"";
```

```
    print $DATA{'WIDTH_TABLE_1'} . "\" height=\"21\" ";
    print "border=\"0\" cellpadding=\"0\" cellspacing=\"0\">\n";
    print "<TR class=\"menu-table\"><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'};
    print "cgi-bin/admin/index.pl\" class=\"menu\">";
    print "На главную</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'} . "cgi-bin/admin/rubr.pl\" ";
    print "class=\"menu\">Рубрикатор</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'} . "cgi-bin/admin/gbook.pl\" ";
    print "class=\"menu\">Гостевая книга</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'} . "cgi-bin/admin/moder.pl\" ";
    print "class=\"menu\">Сайты на модерации</A>\n";
    print "</TD><TD>\n";
    print "<A href=\"" . $DATA{'URL_SITE'};
    print "cgi-bin/admin/catalog.pl\" ";
    print "class=\"menu\">Администрирование каталога</A>\n";
    print "</TD></TR>\n";
    print "</TABLE>\n";
    print "<!-- Панель навигации для администратора. Конец -->\n\n";
}
```

5.6. Структура файла конфигурации

В итоге содержимое файла Allscript.pm должно выглядеть так, как показано в листинге 5.24.

Листинг 5.24. Структура файла Allscript.pm

```
package Allscript;
use strict;
# Экспортируем идентификаторы из модуля
use Exporter;
our @ISA = qw( Exporter );
our @EXPORT = qw( %DATA &f_table_2_start &f_table_2_end
                  &f_table_page_start &f_table_page_end );
```

```
our @EXPORT_OK = qw( &f_passw_generator &f_table_site &f_table_gbook
                                        &f_date_gm );
our %EXPORT_TAGS = (
   "user" => [qw( &f_header_all &f_footer_user )],
   "admin" => [qw( &f_header_all &f_footer_admin &f_menu_admin )]
);
Exporter::export_ok_tags('user', 'admin');

our %DATA;
# URL-адрес сайта (например, http://www.site.ru/)
$DATA{'URL_SITE'} = "http://site.ru/";
$DATA{'WIDTH_TABLE_1'} = "760";  # Ширина таблиц 1-го уровня
$DATA{'WIDTH_TABLE_2'} = "600";  # Ширина таблиц 2-го уровня
$DATA{'WIDTH_TABLE_3'} = "100%";  # Ширина таблиц 3-го уровня

# Данные для подключения к базе данных
$DATA{'HOST'} = "localhost"; # Сервер
$DATA{'LOGIN'} = "root"; # Логин
$DATA{'PASSW'} = ""; # Пароль
$DATA{'DB'} = "CatalogDB"; # База данных

# Данные для писем (например, support <unicross\@mail.ru>)
$DATA{'MAIL_POST'} = "support <unicross\@mail.ru>";
# Для формы обратной связи (например, unicross\@mail.ru)
$DATA{'MAIL_ADRES'} = "unicross\@mail.ru";

# Время жизни сессии
$DATA{'TIME_SESS'} = "+1h";
# Местоположение файлов сессии
$DATA{'TMP_PATH'} = "C:/WebServers/home/site.ru/cgi-bin/user/tmp/";

# Количество сообщений в гостевой книге
$DATA{'count_pos_page_gbook'} = 10;
# Количество сайтов на странице
$DATA{'count_pos_page_site'} = 10;

sub f_style { # Листинг 5.6
   # Таблица стилей
}
```

```
sub f_header { # Листинг 5.5
   # Верхний колонтитул
}
sub f_footer { # Листинг 5.20
   # Нижний колонтитул
}
sub f_logo { # Листинг 5.7
   # Логотип и баннер
}
sub f_menu { # Листинг 5.8
   # Панель навигации
}
sub f_table { # Листинг 5.9
   # Таблица-разделитель
}
sub f_search { # Листинг 5.10
   # Вывод поисковой формы
}
sub f_table_center_start { # Листинг 5.11
   # Таблица для основного содержания страницы. Начало
}
sub f_table_center_end { # Листинг 5.12
   # Таблица для основного содержания страницы. Конец
}
sub f_header_all { # Листинг 5.4
   # Заголовок для всех страниц
}
sub f_footer_user { # Листинг 5.19
   # Выводим нижний колонтитул для пользователей
}
sub f_table_2_start { # Листинг 5.13
   # Таблица второго уровня. Начало
}
sub f_table_2_end { # Листинг 5.14
   # Таблица второго уровня. Конец
}
sub f_search_admin { # Листинг 5.22
   # Поисковая форма для администратора
}
```

```
sub f_menu_admin { # Листинг 5.23
   # Панель навигации для администратора
}
sub f_footer_admin { # Листинг 5.21
   # Выводим нижний колонтитул для администратора
}
sub f_table_site { # Листинг 5.15
   # Описание сайта
}
sub f_table_gbook { # Листинг 5.16
   # Сообщение в гостевой книге
}
sub f_table_page_start { # Листинг 5.17
   # Таблица для количества страниц. Начало
}
sub f_table_page_end { # Листинг 5.18
   # Таблица для количества страниц. Конец
}
sub f_passw_generator { # Листинг 5.25
   # Генератор паролей
}
sub f_date_gm { # Листинг 5.25
   # Форматирование даты
}
1;
```

Все функции из этого файла мы уже рассмотрели, за исключением двух функций: `f_passw_generator()` и `f_date_gm()` (листинг 5.25). Назначение функции для генерации паролей `f_passw_generator()` мы уже обсуждали ранее. Функция `f_date_gm()` предназначена для форматированного вывода даты.

Листинг 5.25. Функции `f_passw_generator()` и `f_date_gm()`

```
sub f_passw_generator {
   # Генератор паролей
   my $count_char = shift();
   $count_char = 8 if (!defined($count_char));
   my @mass = ('a','b','c','d','e','f','g','h','i','j','k','l','m',
   'n','o','p','q','r','s','t','u','v','w','x','y','z',
```

```
    'A','B','C','D','E','F','G','H','I','J','K','L',
    'M','N','O','P','Q','R','S','T','U','V', 'W',
    'X','Y','Z','1','2','3','4','5','6','7','8','9','0');
     my $passw = "";
     for (my $i=0; $i<$count_char; $i++) {
       $passw .= $mass[int(rand(scalar(@mass)))];
     }
    return $passw;
}

sub f_date_gm {
    my @date = gmtime();
    my @day = ("Sun", "Mon", "Tue", "Wed", "Thu",
         "Fri", "Sat");
    my @month = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
          "Aug", "Sep", "Oct", "Nov", "Dec");
    my $year = $date[5] + 1900;
    my $gmdate = "";
    $gmdate = $day[$date[6]];
    if (length($date[3]) == 1) {
       $date[3] = "0" . $date[3];
    }
    $gmdate .= ", " . $date[3] . " ";
    $gmdate .= $month[$date[4]];
    $gmdate .= " " . $year . " ";
    if (length($date[2]) == 1) {
       $date[2] = "0" . $date[2];
    }
    if (length($date[1]) == 1) {
       $date[1] = "0" . $date[1];
    }
    if (length($date[0]) == 1) {
       $date[0] = "0" . $date[0];
    }
    $gmdate .= $date[2] . ":" . $date[1] . ":" . $date[0];
    return $gmdate;
}
```

5.7. Создание SQL-запросов для таблиц

Чтобы иметь возможность тестирования сайта, создадим отдельную базу данных с названием CatalogDB. Открываем программу phpMyAdmin. Для этого в адресной строке Web-браузера набираем **http://localhost/Tools /phpMyAdmin/**. В поле **Создать новую БД** вводим `CatalogDB`. Из списка **Сравнение** выбираем пункт **cp1251_general_ci**. Нажимаем кнопку **Создать**. Или в окне для SQL-запросов набираем команду:

```
CREATE DATABASE CatalogDB DEFAULT CHARACTER SET cp1251
COLLATE cp1251_general_ci;
```

Теперь можно приступать к созданию таблиц. Для доступа к личному кабинету необходима таблица, в которой будут храниться регистрационные данные пользователей — логин (в виде адреса электронной почты) и зашифрованный пароль. В таблице дополнительно создадим поле, с помощью которого можно запретить доступ в систему какому-либо зарегистрированному пользователю. Необходимо также поле с автоматической индексацией и уникальным значением. Идентификатор пользователя будет сохраняться в таблице `site` при регистрации сайта. Кроме того, для ускорения выполнения запросов создадим уникальный индекс по полю логин.

SQL-запрос для создания таблицы приведен в листинге 5.26.

Листинг 5.26. SQL-запрос для таблицы с данными пользователей

```
CREATE TABLE user (
  id_user mediumint(9) auto_increment,
  email char(50),
  passw char(32),
  status_user enum('y','n'),
  PRIMARY KEY (id_user),
  UNIQUE KEY email (email)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Все сайты мы будем распределять по тематике. Для этого необходима таблица с рубрикатором. В таблице создадим два поля: поле с уникальным значением и поле с названием рубрики.

SQL-запрос для создания и заполнения таблицы рубрикатора приведен в листинге 5.27.

Листинг 5.27. SQL-запрос для таблицы рубрикатора

```
CREATE TABLE rubr (
  id_rubr smallint(6) auto_increment,
  name_rubr char(150),
  PRIMARY KEY (id_rubr)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
INSERT INTO rubr (id_rubr, name_rubr) VALUES
(NULL, 'Автомобили'),
(NULL, 'Безопасность'),
(NULL, 'Бытовая техника'),
(NULL, 'Города и регионы'),
(NULL, 'Гостиницы'),
(NULL, 'Знакомства'),
(NULL, 'Игры'),
(NULL, 'Интернет'),
(NULL, 'Искусство'),
(NULL, 'Каталоги'),
(NULL, 'Кино'),
(NULL, 'Компании'),
(NULL, 'Компьютеры'),
(NULL, 'Кулинария'),
(NULL, 'Литература'),
(NULL, 'Магазины On-Line'),
(NULL, 'Мебель'),
(NULL, 'Медицина'),
(NULL, 'Мобильные телефоны'),
(NULL, 'Мода'),
(NULL, 'Музыка'),
(NULL, 'Наука'),
(NULL, 'Недвижимость'),
(NULL, 'Непознанное'),
(NULL, 'Образование'),
(NULL, 'Окна и двери'),
(NULL, 'Подарки и праздники'),
(NULL, 'Политика'),
(NULL, 'Природа'),
(NULL, 'Провайдеры'),
```

```
(NULL, 'Продукты питания'),
(NULL, 'Работа'),
(NULL, 'Развлечения'),
(NULL, 'Разное'),
(NULL, 'Реклама'),
(NULL, 'Сервисы'),
(NULL, 'СМИ'),
(NULL, 'Спорт'),
(NULL, 'Страхование'),
(NULL, 'Строительство'),
(NULL, 'Театр'),
(NULL, 'Товары и услуги'),
(NULL, 'Транспорт'),
(NULL, 'Туризм'),
(NULL, 'Финансовые услуги'),
(NULL, 'Фото'),
(NULL, 'Хобби'),
(NULL, 'Юридические услуги');
```

SQL-запрос для создания таблицы с описаниями сайтов приведен в листинге 5.28.

Листинг 5.28. SQL-запрос для создания таблицы с описаниями сайтов

```
CREATE TABLE site (
  id_site mediumint(9) auto_increment,
  id_rubr smallint(6),
  id_user mediumint(9),
  url_site char(255),
  url_site_dop char(255),
  title char(80),
  deskr text,
  status_site enum('y','n','s'),
  iq_site smallint(6),
  add_date date,
  PRIMARY KEY (id_site),
  KEY id_rubr (id_rubr),
  KEY url_site_dop (url_site_dop)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Таблица содержит следующие поля:

- `id_site` — поле с уникальным значением и автоматической индексацией;
- `id_rubr` — идентификатор рубрики. Должен совпадать с одноименным полем в таблице `rubr`. Для ускорения выполнения запросов создается индекс по этому полю;
- `id_user` — идентификатор пользователя. Должен совпадать с одноименным полем в таблице `user`;
- `url_site` — полный URL-адрес, например, **http://www.site.ru**;
- `url_site_dop` — доменное имя, например, site.ru. Используется для определения уникальности сайта в каталоге. Для ускорения выполнения запросов создается индекс по этому полю;
- `title` — название сайта;
- `deskr` — описание сайта;
- `status_site` — текущий статус. Может принимать следующие значения:
  - `y` — активен;
  - `n` — на модерации;
  - `s` — временно не работает;
- `iq_site` — показатель значимости ресурса в виде числа. По этому полю будет осуществляться сортировка. Чем выше показатель, тем выше сайт в результатах;
- `add_date` — дата регистрации.

Все сообщения пользователей в гостевой книге будут храниться также в базе данных. SQL-запрос для создания таблицы приведен в листинге 5.29.

Листинг 5.29. SQL-запрос для таблицы, используемой гостевой книгой

```
CREATE TABLE gbook (
  id_msg smallint(6) auto_increment,
  author char(55),
  msg text,
  msg_date datetime,
  msg_new enum('y','n'),
  PRIMARY KEY (id_msg)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Поле `msg_new` предназначено для определения статуса сообщения (прочтено или нет).

5.8. Вывод оглавления с количеством в каждом разделе

Прежде чем создать любую программу, необходимо описать алгоритм ее работы. При выводе рубрикатора алгоритм следующий.

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу сценария.
2. Делаем запрос на общее количество зарегистрированных сайтов и сохраняем значение в переменной `$all_count_site`.
3. Делаем запрос на количество зарегистрированных сайтов на текущую дату и сохраняем значение в переменной `$count_site_evr`.
4. Выводим названия рубрик в алфавитном порядке и количеством зарегистрированных сайтов в каждой рубрике независимо от количества. Все названия рубрик разделяем на 3 равных столбца.
5. Выводим значения переменных `$all_count_site` и `$count_site_evr`.
6. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.30.

Листинг 5.30. Содержимое файла /cgi-bin/index.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user );
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Каталог ресурсов Интернета";
# Описание страницы
my $description = "";
```

```
# Ключевые слова для поисковых машин
my $keywords = "Добавить сайт, каталог сайтов";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_user();
   exit();
}
$db->do("SET NAMES cp1251");

my $all_count_site;
my @row_r;
my $res_r = $db->prepare("SELECT COUNT(*) AS s FROM site");
$res_r->execute();
if (!$res_r->err) {
   @row_r = $res_r->fetchrow_array();
   $all_count_site = $row_r[0];
}
$res_r->finish();

my $count_site_evr;
my @row_evr;
my $query_evr = "SELECT COUNT(*) FROM site ";
$query_evr .= "WHERE add_date=CURDATE()";
my $res_evr = $db->prepare($query_evr);
$res_evr->execute();
if (!$res_evr->err) {
   @row_evr = $res_evr->fetchrow_array();
   $count_site_evr = $row_evr[0];
}
$res_evr->finish();
```

```
print "<H1>Каталог ресурсов Интернета</H1><BR>\n";
print "<TABLE align=\"center\" width=\"600\" bgcolor=\"#FFFFFF\" ";
print "border=\"0\" cellpadding=\"2\" cellspacing=\"0\">\n";
print "<TR><TD valign=\"top\">\n";

my $query = "SELECT rubr.id_rubr, rubr.name_rubr, ";
$query .= "COUNT(site.id_site) AS s ";
$query .= "FROM rubr LEFT JOIN site ON rubr.id_rubr=site.id_rubr ";
$query .= "GROUP BY rubr.id_rubr ORDER BY rubr.name_rubr";
my $res = $db->prepare($query);
$res->execute();
if (!$res->err) {
   my $count_rubrikator = $res->rows();
   if ($count_rubrikator>0) {
      my $c1 = $count_rubrikator/3;
      my $c1_1 = int($c1);
      if ($c1 > $c1_1) {
         $c1 = $c1_1 + 1;
      }
      my $c2 = $c1*2;
      my $i = 0;
      while (my @row = $res->fetchrow_array()) {
         if ($i==$c1 || $i==$c2) {
            print "</TD><TD valign=\"top\">\n";
         }
         print "<A href=\"" . $DATA{'URL_SITE'};
         print "cgi-bin/catalog.pl?rubr=$row[0]\">";
         print "$row[1]</A> ($row[2])<BR>\n";
         $i++;
      }
   }
}
$res->finish();

print "</TD></TR>\n";
print "</TABLE>\n";
print "<BR><DIV align=\"center\" class=\"bold\">";
```

```
print "Количество сайтов в каталоге — $all_count_site<BR>\n";
print "Сегодня добавлено — $count_site_evr</DIV>\n";
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();
```

Обратите внимание, что при определении количества сайтов, добавленных за текущий день, мы использовали встроенную функцию MySQL `CURDATE()`, которая возвращает текущую дату в формате `ГГГГ-ММ-ДД`:

```
my $query_evr = "SELECT COUNT(*) FROM site ";
$query_evr .= "WHERE add_date=CURDATE()";
```

Для вывода рубрик в SQL-запросе используется левостороннее объединение, т. к. иначе мы получим только названия рубрик, в которых есть зарегистрированные сайты, и не получим рубрики с нулевым количеством:

```
my $query = "SELECT rubr.id_rubr, rubr.name_rubr, ";
$query .= "COUNT(site.id_site) AS s ";
$query .= "FROM rubr LEFT JOIN site ON rubr.id_rubr=site.id_rubr ";
$query .= "GROUP BY rubr.id_rubr ORDER BY rubr.name_rubr";
```

Объединение выглядит следующим образом:

```
<Таблица1> LEFT JOIN <Таблица2> ON <Таблица1>.<Поле1>=<Таблица2>.<Поле2>
```

Так как названия полей в таблицах одинаковые, то вместо инструкции `ON` можно использовать инструкцию `USING`:

```
<Таблица1> LEFT JOIN <Таблица2> USING (<Поле>)
```

Например:

```
my $query = "SELECT rubr.id_rubr, rubr.name_rubr, ";
$query .= "COUNT(site.id_site) AS s ";
$query .= "FROM rubr LEFT JOIN site USING (id_rubr) ";
$query .= "GROUP BY rubr.id_rubr ORDER BY rubr.name_rubr";
```

Для разделения рубрик на 3 равных столбца делим общее количество рубрик на 3 и округляем до целого числа:

```
my $c1 = $count_rubrikator/3;
my $c1_1 = int($c1);
if ($c1 > $c1_1) {
   $c1 = $c1_1 + 1;
}
```

Для получения второго числа умножаем на 2:

```
my $c2 = $c1*2;
```

Далее проверяем условие. Если текущее значение счетчика равняется первому или второму числу, то выводим теги для разделения ячеек таблицы:

```
if ($i==$c1 || $i==$c2) {
   print "</TD><TD valign=\"top\">\n";
}
```

Каждое название рубрики является ссылкой, в которой передается идентификатор рубрики методом `GET`. По этому идентификатору будут выводиться сайты, зарегистрированные в указанной рубрике:

```
print "<A href=\"" . $DATA{'URL_SITE'};
print "cgi-bin/catalog.pl?rubr=$row[0]\">";
print "$row[1]</A> ($row[2])<BR>\n";
```

5.9. Вывод каталога по параметрам

При выводе каталога алгоритм работы программы следующий.

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу сценария.
2. С помощью функции `param()` из модуля `CGI` получаем параметры, переданные в строке запроса, и сохраняем их в переменных `$rubr` (идентификатор рубрики) и `$page` (номер страницы).
3. С помощью регулярного выражения проверяем, чтобы идентификатор рубрики был числом. Если это не так, то идентификатору присваивается значение 1.
4. Делаем запрос для определения названия рубрики и выводим ее название в заголовок страницы.
5. Делаем запрос на общее количество сайтов в рубрике.
6. Проверяем формат номера страницы и рассчитываем начальную позицию.
7. Делаем запрос на выборку заданного количества сайтов, начиная с начальной позиции.
8. Выводим описания сайтов.
9. Если число сайтов больше заданного максимального количества сайтов на странице, то выводим номера страниц.
10. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.31.

Листинг 5.31. Содержимое файла /cgi-bin/catalog.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_table_site );

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "Content-type: text/html; charset=windows-1251\n\n";
   print "<DIV style=\"font-size: 16px; font-weight: bold; ";
   print "text-align: center; color: #FF0000\">";
   print "Не удалось установить соединение с базой данных</DIV>";
   exit();
}
$db->do("SET NAMES cp1251");
print "Content-type: text/html; charset=windows-1251\n\n";
my $rubr = param('rubr') || 0;
my $page = param('page') || 1;
if ($rubr !~ /^[0-9]+$/ || $rubr == 0) {
   $rubr = 1;
}
my $rubrika = "";
my $res_r = $db->prepare("SELECT * FROM rubr WHERE id_rubr=$rubr");
$res_r->execute();
```

```
if (!$res_r->err && $res_r->rows() > 0) {
   my @row_r = $res_r->fetchrow_array();
   $rubrika = $row_r[1];
}
$res_r->finish();

# Заголовок
my $title = "Каталог сайтов >> Рубрика — $rubrika";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "$rubrika";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);
&f_table_2_start(); # Выводим таблицу второго уровня
print "<H1>Каталог сайтов >> $rubrika</H1><BR>\n";

# Определяем количество сайтов в рубрике
my $count = 0;
my $query_count = "SELECT COUNT(*) FROM site ";
$query_count .= "WHERE status_site='y' AND id_rubr=$rubr";
my $res_count = $db->prepare($query_count);
$res_count->execute();
if (!$res_count->err && $res_count->rows() == 1) {
   my @row_count = $res_count->fetchrow_array();
   $count = $row_count[0];
}
$res_count->finish();

# Выводим сайты
if ($count =~ /^[0-9]+$/ && $count > 0) {
   $page = 1 if (!defined($page) || $page < 1);
   $page = 1 if ($page !~ /^[0-9]+$/);
   my $count_page = $count/$DATA{'count_pos_page_site'};
   my $count_page2 = int($count_page);
   if ($count_page > $count_page2) {
      $count_page = $count_page2 + 1;
   }
```

```
   $page = $count_page if ($page > $count_page);
   my $pos_start = ($page - 1) * $DATA{'count_pos_page_site'};
   my $query = "SELECT * FROM site WHERE status_site='y' ";
   $query .= "AND id_rubr=$rubr ";
   $query .= "ORDER BY iq_site desc ";
   $query .= "LIMIT $pos_start, $DATA{'count_pos_page_site'}";
   my $res = $db->prepare($query);
   $res->execute();
   if (!$res->err && $res->rows() > 0) {
      while (my @row = $res->fetchrow_array()) {
        # Выводим описание
        &f_table_site($row[3], $row[5], $row[6]);
      }
   }
   $res->finish();

   # Выводим количество страниц
   if ($count > $DATA{'count_pos_page_site'}) {
      &f_table_page_start(); # Таблица для количества страниц Начало
      print "<SPAN class=\"bold\">Страницы:</SPAN>\n";
      for (my $j = 1; $j < $count_page + 1; $j++) {
         if ($j == $page) {
            print "[$j] \n";
         }
         else {
            print "<A href=\"?rubr=$rubr&page=$j\">$j</A> \n";
         }
      }
      &f_table_page_end(); # Таблица для количества страниц Конец
   }
}
else {
   print "<DIV align=\"center\" class=\"bold\">Сайтов нет</DIV><BR>\n";
}

&f_table_2_end(); # Конец таблицы второго уровня
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();
```

5.10. Организация поиска по каталогу

При поиске по каталогу алгоритм работы программы следующий.

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу сценария.
2. С помощью функции `param()` из модуля `CGI` получаем параметры, переданные в строке запроса, и сохраняем их в переменных `$text` (поисковая фраза) и `$page` (номер страницы).
3. Удаляем все спецсимволы из поисковой фразы и проверяем ее на допустимость.
4. Для вывода поисковой фразы в строку поисковой формы создаем переменную `$text2`, в которой все кавычки заменяем на HTML-эквиваленты.
5. Добавляем защитные слэши перед спецсимволами в поисковой фразе.
6. При поиске по шаблону символы "_" и "%" являются специальными. По этой причине добавляем перед ними защитные слэши.
7. Делаем запрос на выборку.
8. Проверяем корректность номера страницы и рассчитываем начальную и конечную позицию.
9. Выводим описания сайтов.
10. Если число сайтов больше заданного максимального количества сайтов на странице, то выводим номера страниц.
11. Если существуют ошибки, то выводим их описание.
12. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.32.

Листинг 5.32. Содержимое файла /cgi-bin/search.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
```

```
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_date_gm &f_table_site );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_user();
   exit();
}
$db->do("SET NAMES cp1251");

my $err_text = "";
my $text = param('search') || "";
my $page = param('page') || 1;
$text =~ s/&/&amp;/g;
$text =~ s/</&lt;/g;
$text =~ s/>/&gt;/g;
$text =~ s/\r//g;
$text =~ s/\n/ /g;
$text =~ s/\t/ /g;
$text =~ s/\s*$//;
$text =~ s/^\s*//;
if ($text eq "") {
   $err_text .= "Не задана строка поиска<BR>";
}
if (length($text) < 3) {
   $err_text .= "В поле допустимо не менее 3 символов<BR>";
}
```

```
if (length($text) > 50) {
   $err_text .= "В поле допустимо не более 50 символов<BR>";
}
$text =~ s/"/&quot;/g;
my $text2 = $text;
# Заголовок
my $title = "Результаты поиска — $text2";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2, $text2);
&f_table_2_start(); # Выводим таблицу второго уровня
print "<H1>Результаты поиска</H1><BR>\n";
if($err_text eq "") {
   # Добавляем защитные слэши перед спецсимволами
   $text =~ s/\\/\\\\/g;
   $text =~ s/'/\\'/g;
   $text =~ s/%/\\%/g;
   $text =~ s/_/\\_/g;
   # Выводим результаты поиска
   my $query = "SELECT * FROM site ";
   $query .= "WHERE status_site='y' AND (title LIKE '%$text%' ";
   $query .= "OR deskr LIKE '%$text%' ";
   $query .= "OR url_site LIKE '%$text%') ";
   $query .= "ORDER BY iq_site DESC";
   # Выполняем запрос
   my $count = 0;
   my $all_array;
   my $result = $db->prepare($query);
   $result->execute();
   if (!$result->err) {
      $all_array = $result->fetchall_arrayref();
      $count = scalar(@$all_array);
   }
   # Выводим сайты
   my $count_page = 0;
```

```
   if ($count =~ /^[0-9]+$/ && $count > 0) {
      $page = 1 if (!defined($page) || $page < 1);
      $page = 1 if ($page !~ /^[0-9]+$/);
      $count_page = $count/$DATA{'count_pos_page_site'};
      my $count_page2 = int($count_page);
      if ($count_page > $count_page2) {
         $count_page = $count_page2 + 1;
      }
      $page = $count_page if ($page > $count_page);
      my $pos_end = $page * $DATA{'count_pos_page_site'};
      my $pos_start = $pos_end - $DATA{'count_pos_page_site'};
      $pos_end = $count if ($pos_end > $count);
      $pos_start = 0 if ($pos_start < 0);
      for (my $i = $pos_start; $i < $pos_end; $i++) {
         my $url_site = $all_array->[$i]->[3];
         my $title_site = $all_array->[$i]->[5];
         my $deskr_site = $all_array->[$i]->[6];
         &f_table_site($url_site, $title_site, $deskr_site);
      }
      # Выводим количество страниц
      if ($count > $DATA{'count_pos_page_site'}) {
         &f_table_page_start(); # Таблица для количества страниц Начало
         print "<SPAN class=\"bold\">Страницы:</SPAN>\n";
         for (my $j = 1; $j < $count_page + 1; $j++) {
            if ($j == $page) { print "[$j] \n"; }
            else {
               print "<A href=\"?page=$j&search=$text2\">$j</A> \n";
            }
         }
         &f_table_page_end(); # Таблица для количества страниц Конец
      }
   }
   else {
      print "<DIV align=\"center\" class=\"bold\">";
      print "По вашему запросу ничего не найдено</DIV><BR>\n";
   }
}
```

```
if ($err_text ne "") {
   print "<DIV class=\"err\">$err_text</DIV><BR>\n";
}
&f_table_2_end(); # Конец таблицы второго уровня
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();
```

5.11. Создание гостевой книги

Созданная книга должна содержать следующие возможности:

- проверку корректности введенных данных с использованием JavaScript на стороне клиента;
- проверку корректности введенных данных на стороне сервера с помощью Perl;
- на одной странице должно быть не более заданного количества сообщений, а снизу должны быть ссылки на все остальные страницы;
- при нажатии на кнопку **Обновить** сообщение не должно повторно добавляться в базу;
- одно сообщение может состоять не более чем из 100 строк. Причем подряд должно быть не более двух пустых строк;
- не должно быть более 25 символов подряд без пробела;
- все HTML-теги и спецсимволы должны быть заменены на HTML-эквиваленты;
- в случае ошибки должно быть выведено соответствующее сообщение.

Алгоритм работы программы следующий.

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. С помощью функции `param()` из модуля `CGI` получаем параметры, переданные скрипту, и сохраняем их в переменных `$page` (номер страницы), `$author` (автор) и `$msg` (текст сообщения).
3. Если форма отправлена (параметр `param('go')` будет существовать):
   - заменяем все теги и специальные символы на HTML-эквиваленты;
   - проверяем значения переменных на допустимость;

- проверяем на повтор, сравнивая новое сообщение с последним добавленным;
- экранируем все специальные символы с помощью защитных слэшей;
- добавляем новое сообщение в базу данных.

4. Делаем запрос на общее количество сообщений в гостевой книге.
5. Проверяем корректность номера страницы и рассчитываем начальную и конечную позицию.
6. Делаем запрос на выборку и выводим сообщения.
7. Если число сообщений больше заданного максимального количества сообщений на странице, то выводим номера страниц.
8. Если существуют ошибки, то выводим их описание.
9. Выводим форму. Если возникли ошибки при добавлении сообщения, то заполняем все поля для редактирования.
10. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.33.

Листинг 5.33. Содержимое файла /cgi-bin/gbook.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_table_gbook &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";
```

```
# Заголовок
my $title = "Гостевая книга";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_user();
   exit();
}
$db->do("SET NAMES cp1251");

print <<SCRIPT_TEXT;
<SCRIPT language="JavaScript">
<!--
function f_submit() {
   var frm = document.formGbook;
   if (frm.author.value=="" || frm.msg.value=="") {
      window.alert("Не заполнено обязательное поле");
      return false;
   }
   if (frm.author.value.length>50) {
      window.alert("В поле Имя допустимо не более 50 символов");
      frm.author.focus();
      return false;
   }
   if (frm.msg.value.length>2000) {
      window.alert("В поле Сообщение допустимо не более 2000 символов");
      frm.msg.focus();
```

```
      return false;
   }
return true;
}
//-->
</SCRIPT>
SCRIPT_TEXT

&f_table_2_start(); # Выводим таблицу второго уровня
print "<H1>Гостевая книга</H1><BR>\n";
my $page = param('page') || 1;
my $author = param('author') || "";
my $msg = param('msg') || "";
my $err_msg = "";
if (defined(param('go'))) { # Форма отправлена
   # Заменяем спецсимволы
   $author =~ s/&/&amp;/g;
   $author =~ s/</&lt;/g;
   $author =~ s/>/&gt;/g;
   $author =~ s/"/&quot;/g;
   $msg =~ s/&/&amp;/g;
   $msg =~ s/</&lt;/g;
   $msg =~ s/>/&gt;/g;
   $msg =~ s/"/&quot;/g;
   # Удаляем пробельные символы
   {
      $author =~ s/\r//g;
      $msg =~ s/\r//g;
      local $/ = "";
      chomp($author);
      chomp($msg);
   }
   my @mass = split(/\n/, $msg);
   if (scalar(@mass) <= 100) {
      $msg =~ s/\n/<BR>/g;
   }
   else {
      $err_msg .= "В сообщении допустимо не более 100 строк!<BR>\n";
   }
```

```
$author =~ s/\\/\\\\/g;
$msg =~ s/\\/\\\\/g;
if (length($author) > 50 || length($author) == 0) {
   $err_msg .= "Поле имя не заполнено или длина более ";
   $err_msg .= "50 символов!<BR>\n";
}
if (length($msg) > 2000 || length($msg) == 0) {
   $err_msg .= "Поле сообщение не заполнено или длина ";
   $err_msg .= "более 2000 символов!<BR>\n";
}
if ($author =~ /[a-za-я0-9,\.\"\']{26}/i) {
   $err_msg .= "Поле имя содержит более 25 ";
   $err_msg .= "символов без пробела!<BR>\n";
}
if ($msg =~ /[a-za-я0-9,\.\"\']{26}/i) {
   $err_msg .= "Поле сообщение содержит более 25 ";
   $err_msg .= "символов без пробела!<BR>\n";
}
if ($msg =~ /<BR><BR><BR>/) {
   $err_msg .= "Поле сообщение содержит более 2 символов ";
   $err_msg .= "перевода строки подряд!<BR>\n";
}
if ($err_msg eq "") { # Если ошибок нет
   # Добавляем защитные слэши перед спецсимволами
   $author =~ s/'/\\'/g;
   $msg =~ s/'/\\'/g;
   my $query = "INSERT INTO gbook VALUES ";
   $query .= "(NULL, '$author', '$msg', NOW(), 'n')";
   # Проверяем на повтор
   my $query2 = "SELECT * FROM gbook ORDER BY id_msg DESC LIMIT 0,1";
   my $test_author;
   my $test_msg;
   my $result = $db->prepare($query2);
   $result->execute();
   if (!$result->err) {
      if ($result->rows() == 1) {
         my @row = $result->fetchrow_array();
         $test_author = $row[1];
```

```
            $test_msg = $row[2];
            $test_author =~ s/\\/\\\\/g;
            $test_msg =~ s/\\/\\\\/g;
            $test_author =~ s/'/\\'/g;
            $test_msg =~ s/'/\\'/g;
            if ($test_author eq $author && $test_msg eq $msg) {
            }
            else {
               # Добавляем новое сообщение
               $db->do($query);
            }
            $page = 1;
         }
         elsif ($result->rows() == 0) {
            # Добавляем первое сообщение
            $db->do($query);
         }
      }
      $result->finish();
      $author = "";
      $msg = "";
   }
}

# Определяем количество сообщений
my $count = 0;
my $query_count = "SELECT COUNT(*) FROM gbook";
my $res_count = $db->prepare($query_count);
$res_count->execute();
if (!$res_count->err && $res_count->rows() == 1) {
   my @row_count = $res_count->fetchrow_array();
   $count = $row_count[0];
}
$res_count->finish();

# Выводим сообщения
if ($count =~ /^[0-9]+$/ && $count > 0) {
   $page = 1 if (!defined($page) || $page < 1);
   $page = 1 if ($page !~ /^[0-9]+$/);
```

```
    my $count_page = $count/$DATA{'count_pos_page_gbook'};
    my $count_page2 = int($count_page);
    if ($count_page > $count_page2) {
       $count_page = $count_page2 + 1;
    }
    $page = $count_page if ($page > $count_page);
    my $pos_start = ($page - 1) * $DATA{'count_pos_page_gbook'};
    my $q = "SELECT author, msg, ";
    $q .= "DATE_FORMAT(msg_date, '%d.%m.%Y %H:%i') AS s ";
    $q .= "FROM gbook ORDER BY id_msg DESC ";
    $q .= "LIMIT $pos_start, $DATA{'count_pos_page_gbook'}";
    my $res = $db->prepare($q);
    $res->execute();
    if (!$res->err && $res->rows() > 0) {
       while (my @row = $res->fetchrow_array()) {
         # Выводим описание
         &f_table_gbook($row[2], $row[0], $row[1]);
       }
    }
    else { print "Ошибка"; }
    $res->finish();

    # Выводим количество страниц
    if ($count > $DATA{'count_pos_page_gbook'}) {
       &f_table_page_start(); # Таблица для количества страниц Начало
       print "<SPAN class=\"bold\">Страницы:</SPAN>\n";
       for (my $j = 1; $j < $count_page + 1; $j++) {
          if ($j == $page) {
             print "[$j] \n";
          }
          else {
             print "<A href=\"?page=$j\">$j</A> \n";
          }
       }
       &f_table_page_end(); # Таблица для количества страниц Конец
    }
}
```

```
else {
   print "<DIV align=\"center\" class=\"bold\">";
   print "Сообщений нет</DIV><BR>\n";
}

&f_table_2_end(); # Конец таблицы второго уровня
print "<DIV align=\"center\"><HR width=\"70%\">\n";
print "<H1>Новое сообщение</H1>\n";

if ($err_msg) {
   # Если возникли ошибки, то выводим их описание
   $author =~ s/\\\\/\\/g;
   $msg =~ s/\\\\/\\/g;
   $msg =~ s/<BR>/\n/g;
   print "<DIV class=\"err\">Сообщение не добавлено ";
   print "по сл. причинам:<BR><BR>\n";
   print "$err_msg</DIV><BR>\n";
}

print <<FORM_TEXT;
<FORM method="POST" name="formGbook" id="formGbook"
onsubmit="return f_submit();">
<SPAN class="bold">Ваше имя:</SPAN>
<INPUT type="text" class="txt_frm" style="width: 260px"
name="author" id="author" value="$author">
<SPAN class="bold">Сообщение:</SPAN><BR>
<TEXTAREA name="msg" id="msg" class="textarea_frm"
style="width: 500px; height: 150px">$msg
</TEXTAREA><BR>
<INPUT type="submit" name="go" value="Добавить сообщение">
</FORM></DIV>
FORM_TEXT
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();
```

Мы преобразуем дату из формата MySQL в требуемый формат с помощью функции DATE_FORMAT():

```
my $q = "SELECT author, msg, ";
$q .= "DATE_FORMAT(msg_date, '%d.%m.%Y %H:%i') AS s ";
$q .= "FROM gbook ORDER BY id_msg DESC ";
$q .= "LIMIT $pos_start, $DATA{'count_pos_page_gbook'}";
```

При добавлении нового сообщения в базу данных используем встроенную функцию MySQL NOW():

```
my $query = "INSERT INTO gbook VALUES ";
$query .= "(NULL, '$author', '$msg', NOW(), 'n')";
```

5.12. Создание формы обратной связи

Алгоритм работы программы следующий:

1. Если форма отправлена (параметр param('go') будет существовать):
   - с помощью функции param() из модуля CGI получаем параметры, переданные скрипту, и сохраняем их в переменных $emails (E-mail-адрес), $tema (тема сообщения) и $msg (текст сообщения);
   - проверяем значения переменных на допустимость;
   - если ошибок нет, то отправляем письмо.
2. Если существуют ошибки, то выводим их описание.
3. Если существует сообщение об удачной отправке письма, то выводим его.
4. Отображаем форму. Если возникли ошибки при отправке сообщения, то заполняем все поля для редактирования.

Исходный код программы приведен в листинге 5.34.

Листинг 5.34. Содержимое файла /cgi-bin/contact.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use CGI qw( :standard );
```

```
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user );

print "Content-type: text/html; charset=windows-1251\n\n";
# Заголовок
my $title = "Форма обратной связи";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

print <<SCRIPT_COD;
<SCRIPT language="JavaScript">
<!--
function f_test_mail(m_Mail) {
   c_Reg = /^[a-z0-9_\\.\\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}\$/i;
   if (!c_Reg.test(m_Mail)) return false;
   return true;
}
function f_submit() {
   var m_f = document.i_form;
   if (m_f.i_msg.value=="" || m_f.i_tema.value=="") {
      window.alert("Не заполнено обязательное поле");
      return false;
   }
   if (m_f.i_tema.value.length>50) {
      window.alert("В поле Тема допустимо не более 50 символов");
      m_f.i_tema.focus();
      return false;
   }
  if (m_f.i_emails.value.length>50 || !f_test_mail(m_f.i_emails.value)) {
      window.alert("Недопустимое значение поля E-mail");
      m_f.i_emails.focus();
```

```
      return false;
  }
return true;
}
//-->
</SCRIPT>
SCRIPT_COD
&f_table_2_start(); # Выводим таблицу второго уровня
print "<H1>Обратная связь</H1><BR>\n";

my $emails = "";
my $tema = "";
my $msg = "";
my $err_msg = "";
my $ok_msg = "";

if (defined(param('go'))) {
   # Форма отправлена
   $emails = param('i_emails') || "";
   $tema = param('i_tema') || "";
   $msg = param('i_msg') || "";
   my $pattern = qr/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/;
   if ($emails !~ /$pattern/i) {
      $err_msg .= "Недопустимый адрес E-mail !!!<BR>";
   }
   if (length($emails) > 50) {
      $err_msg .= "Длина E-mail больше допустимой!!!<BR>";
   }
   if (length($tema) > 50 || length($tema) == 0) {
      $err_msg .= "Поле Тема не заполнено или длина более 50 ";
      $err_msg .= "символов!<BR>\n";
   }
   if (length($msg) == 0) {
      $err_msg .= "Поле Сообщение не заполнено!<BR>\n";
   }
   if ($err_msg eq "") {
      # Если ошибок нет, отправляем письмо
      my $header = "From: $emails\n";
```

```
      $header .= "To: " . $DATA{'MAIL_ADRES'} . "\n";
      $header .= "Subject: $tema\n";
      $header .= "Content-Type: text/plain; charset=windows-1251\n";
      $header .= "Content-Transfer-Encoding: 8bit\n\n";
      open(MAIL, "| /usr/sbin/sendmail -t") or die("Ошибка $!");
      print MAIL $header;
      print MAIL $msg;
      close(MAIL) or die("Ошибка $!");
      $emails = $tema = $msg = "";
      $ok_msg = "Ваше сообщение успешно отправлено<BR>";
   }
}
if ($err_msg ne "") {
   # Если возникли ошибки, то выводим их описание
   $tema =~ s/&/&amp;/g;
   $tema =~ s/</&lt;/g;
   $tema =~ s/>/&gt;/g;
   $tema =~ s/"/&quot;/g;
   $msg =~ s/&/&amp;/g;
   $msg =~ s/</&lt;/g;
   $msg =~ s/>/&gt;/g;
   print "<DIV class=\"err\">Сообщение не отправлено по сл. ";
   print "причинам:<BR><BR>\n";
   print "$err_msg<DIV><BR>\n";
}
if ($ok_msg ne "") {
   print "<DIV class=\"ok\">$ok_msg</DIV><BR>\n";
}
print <<FORM_COD;
<FORM method="POST" name="i_form" id="i_form"
onsubmit="return f_submit();">
<TABLE width="100%" border="0" cellpadding="1" align="center">
<TR><TD align="right" width="40%">
<SPAN class="bold">Ваш E-mail: </SPAN>
</TD><TD>
<INPUT type="text" name="i_emails" id="i_emails" maxlength="50" size="40"
value="$emails" class="txt_frm">
</TD></TR>
```

```
<TR><TD align="right" width="40%">
<SPAN class="bold">Тема: </SPAN>
</TD><TD>
<INPUT type="text" name="i_tema" id="i_tema" size="40" maxlength="50"
class="txt_frm" value="$tema">
</TD></TR>
<TR><TD align="right" width="40%"><SPAN class="bold">Сообщение:</SPAN>
</TD><TD>
<TEXTAREA name="i_msg" id="i_msg" class="textarea_frm"
style="width: 300px; height: 150px">$msg</TEXTAREA>
</TD></TR>
<TR><TD align="right" width="40%">.:</TD><TD>
<INPUT type="submit" name="go" value="Отправить" class="txt_frm">
</TD></TR>
</TABLE>
</FORM><BR>
FORM_COD

&f_table_2_end(); # Конец таблицы второго уровня
# Выводим нижний колонтитул
&f_footer_user();
```

5.13. Создание страниц регистрации ошибок

Для регистрации и вывода сообщений об ошибках 401, 403 и 404 создадим отдельные страницы. В случае возникновения ошибки сообщение о ней будет записываться в файл, а пользователь получит сообщение в дизайне сайта.

Содержимое файла err401.pl приведено в листинге 5.35.

Листинг 5.35. Содержимое файла /cgi-bin/err401.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
```

```
use strict;
use POSIX;
# Настройка локали
use locale;
setlocale(LC_ALL, 'ru_RU.CP1251');
use Fcntl qw( :flock );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Ошибка 401";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

print "<H1>Ошибка 401</H1><BR>\n";

open(FILE, '>>err401.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
my $date = strftime("%H:%M:%S %d.%m.%Y", localtime);
my $remoteUser = $ENV{'REMOTE_USER'} || "";
my $requestUri = $ENV{'REQUEST_URI'} || "";
my $httpReferer = $ENV{'HTTP_REFERER'} || "";
my $userAgent = $ENV{'HTTP_USER_AGENT'} || "";
my $remoteAddr = $ENV{'REMOTE_ADDR'} || "";
print FILE "$date\nПользователь: $remoteUser";
```

```
print FILE "\nТекст ошибки: $requestUri";
print FILE "\nСсылка: $httpReferer";
print FILE "\nБраузер: $userAgent";
print FILE "\nIP-адрес: $remoteAddr";
print FILE "\n\n";
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);

# Выводим нижний колонтитул
&f_footer_user();
```

Содержимое файла err403.pl приведено в листинге 5.36.

Листинг 5.36. Содержимое файла /cgi-bin/err403.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use POSIX;
# Настройка локали
use locale;
setlocale(LC_ALL, 'ru_RU.CP1251');
use Fcntl qw( :flock );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";
```

```
# Заголовок
my $title = "Ошибка 403";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

print "<H1>Ошибка 403</H1><BR>\n";

open(FILE, '>>err403.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
my $date = strftime("%H:%M:%S %d.%m.%Y", localtime);
my $remoteUser = $ENV{'REMOTE_USER'} || "";
my $requestUri = $ENV{'REQUEST_URI'} || "";
my $httpReferer = $ENV{'HTTP_REFERER'} || "";
my $userAgent = $ENV{'HTTP_USER_AGENT'} || "";
my $remoteAddr = $ENV{'REMOTE_ADDR'} || "";
print FILE "$date\nПользователь: $remoteUser";
print FILE "\nТекст ошибки: $requestUri";
print FILE "\nСсылка: $httpReferer";
print FILE "\nБраузер: $userAgent";
print FILE "\nIP-адрес: $remoteAddr";
print FILE "\n\n";
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);

# Выводим нижний колонтитул
&f_footer_user();
```

Содержимое файла err404.pl приведено в листинге 5.37.

Листинг 5.37. Содержимое файла /cgi-bin/err404.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>errlog.txt") or die("Ошибка\n");
```

```
   carpout(\*ERRLOG);
}
use strict;
use POSIX;
# Настройка локали
use locale;
setlocale(LC_ALL, 'ru_RU.CP1251');
use Fcntl qw( :flock );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Ошибка 404";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

print "<H1>Ошибка 404</H1><BR>\n";

open(FILE, '>>err404.txt') or die("Ошибка $!");
flock(FILE, LOCK_EX) or die("Ошибка $!");
my $date = strftime("%H:%M:%S %d.%m.%Y", localtime);
my $remoteUser = $ENV{'REMOTE_USER'} || "";
my $requestUri = $ENV{'REQUEST_URI'} || "";
my $httpReferer = $ENV{'HTTP_REFERER'} || "";
my $userAgent = $ENV{'HTTP_USER_AGENT'} || "";
my $remoteAddr = $ENV{'REMOTE_ADDR'} || "";
```

```
print FILE "$date\nПользователь: $remoteUser";
print FILE "\nТекст ошибки: $requestUri";
print FILE "\nСсылка: $httpReferer";
print FILE "\nБраузер: $userAgent";
print FILE "\nIP-адрес: $remoteAddr";
print FILE "\n\n";
flock(FILE, LOCK_UN) or die("Ошибка $!");
close(FILE);

# Выводим нижний колонтитул
&f_footer_user();
```

Для того чтобы эти страницы отображались при соответствующих ошибках, необходимо создать в папке файл .htaccess и добавить строки, приведенные в листинге 5.38.

Листинг 5.38. Содержимое файла /cgi-bin/.htaccess

```
ErrorDocument 401 /cgi-bin/err401.pl
ErrorDocument 403 /cgi-bin/err403.pl
ErrorDocument 404 /cgi-bin/err404.pl
<Files *.txt>
   Deny from all
</Files>
<Files *.pm>
   Deny from all
</Files>
<Files cgisess*>
   Deny from all
</Files>
```

5.14. Создание Личного кабинета для пользователей при помощи Perl

Для добавления сайтов в каталог создадим Личный кабинет. Это позволит защититься от автоматического добавления сайтов. В дальнейшем можно добавить возможность редактирования описания сайтов пользователями и другие возможности.

Для реализации Личного кабинета создадим каталог user, в котором разместим три файла:

- index.pl — для входа в систему, регистрации новых пользователей и восстановления пароля;
- add.pl — для регистрации новых сайтов;
- exit.pl — для выхода из системы.

5.14.1. Создание страницы для входа в Личный кабинет

Данный пример демонстрирует возможность обработки нескольких форм с помощью одного файла.

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. Если форма для входа в систему отправлена (параметр `param('enter')` будет существовать):
   - с помощью функции `param()` из модуля `CGI` получаем параметры, переданные скрипту, и сохраняем их в переменных `$login` (логин) и `$passw` (пароль);
   - проверяем корректность введенных данных;
   - шифруем введенный пароль с помощью функции `md5_hex()`;
   - делаем запрос. Если пользователь найден, то запускаем сессию и регистрируем в ней переменные, а затем перебрасываем пользователя на страницу добавления сайта;
   - завершаем работу скрипта.
3. Если форма для регистрации нового пользователя отправлена (параметр `param('add')` существует), выполняем процедуры:
   - с помощью функции `param()` из модуля `CGI` получаем параметры, переданные скрипту, и сохраняем их в переменных `$emails` (E-mail-адрес) и `$pass` (пароль);
   - проверяем корректность введенных данных;
   - проверяем, не зарегистрирован ли E-mail раньше;
   - шифруем введенный пароль с помощью функции `md5_hex()`;
   - регистрируем пользователя;

- запускаем сессию и регистрируем в ней переменные, а затем перебрасываем пользователя на страницу добавления сайта;
- завершаем работу скрипта.

4. Если форма для восстановления пароля отправлена (параметр `param('mail_pass')` будет существовать), выполняем процедуры:
   - с помощью функции `param()` из модуля `CGI` получаем E-mail-адрес и сохраняем его в переменной `$mails`;
   - проверяем корректность введенных данных;
   - проверяем, зарегистрирован ли E-mail раньше;
   - генерируем новый пароль с помощью функции `f_passw_generator()`;
   - шифруем пароль с помощью функции `md5_hex()`;
   - добавляем новый пароль в базу данных;
   - отправляем новый пароль на E-mail.
5. Если возникли ошибки при входе в систему, то выводим их описание.
6. Выводим форму "Вход в систему".
7. Если возникли ошибки при регистрации, то выводим их описание.
8. Выводим форму "Регистрация нового пользователя".
9. Если возникли ошибки при восстановлении пароля, то выводим их описание.
10. В случае успешной отправки нового пароля на E-mail выводим подтверждение.
11. Выводим форму "Восстановить пароль".
12. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.39.

Листинг 5.39. Содержимое файла /cgi-bin/user/index.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
```

```
use CGI qw( :standard );
use CGI::Session;
use Digest::MD5 qw( md5_hex );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_passw_generator );

my $err_enter = "";
my $err_add = "";
my $err_mail = "";
my $msg = "";
my $emails = "";
my $pass = "";
my $login = "";

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "Content-type: text/html; charset=windows-1251\n\n";
   print "<DIV style=\"font-size: 16px; font-weight: bold; ";
   print "text-align: center; color: #FF0000\">";
   print "Не удалось установить соединение с базой данных</DIV>";
   exit();
}
$db->do("SET NAMES cp1251");

# Вход в Личный кабинет
if (defined(param('enter'))) {
   $login = param("login") || "";
   my $passw = param("passw") || "";
   $passw =~ s/\s*$//;
   $passw =~ s/^\s*//;
   my $pattern = qr/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/;
   if ($login !~ /$pattern/i) {
      $err_enter = "Недопустимый адрес E-mail !!!<BR>";
   }
```

```
if (length($login) > 50) {
   $err_enter .= "Длина E-mail больше допустимой!!!<BR>";
}
if (length($passw) > 16 || length($passw) < 6) {
   $err_enter .= "Длина пароля должна быть в пределах ";
   $err_enter .= "от 6 до 16 символов.<BR>";
}
if ($passw !~ /^[a-z0-9_\-]{6,16}$/i) {
   $err_enter .= "В пароле допустимы только буквы ";
   $err_enter .= "A-Z (a-z) или цифры 0-9<BR>";
}
if ($err_enter eq "") { # Если ошибок нет
   $passw = md5_hex($passw);
   my $query_enter = "SELECT * FROM user ";
   $query_enter .= "WHERE email='$login' AND passw='$passw'";
   my $res_enter = $db->prepare($query_enter);
   $res_enter->execute();
   if (!$res_enter->err) {
      if ($res_enter->rows() == 1) {
         my $sess = CGI::Session->new("driver:file", undef,
                          {Directory=>$DATA{'TMP_PATH'}})
                          or die CGI::Session->errstr();
         $sess->name('SID');
         my $cookies = cookie(-name=>$sess->name(),
                              -value=>$sess->id(), -path=>'/');
         print "Set-Cookie: $cookies\n";
         $sess->param(-name=>'Login', -value=>$login);
         $sess->param(-name=>'Password', -value=>$passw);
         $sess->expire($DATA{'TIME_SESS'});
         $sess->flush();
         print "Location: add.pl\n\n";
         exit();
      }
      else {
         $err_enter .= "Логин/пароль не найден.<BR>";
      }
   }
```

```
      else {
         $err_enter .= "Произошла ошибка.<BR>";
         $err_enter .= "Попробуйте сделать запрос ";
         $err_enter .= "через некоторое время<BR>";
      }
   }
}
# Регистрация нового пользователя
if (defined(param('add'))) {
   $emails = param('emails') || "";
   $pass = param('pass') || "";
   my $pattern = qr/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/;
   if ($emails !~ /$pattern/i) {
      $err_add = "Недопустимый адрес E-mail !!!<BR>";
   }
   if (length($emails) > 50) {
      $err_add .= "Длина E-mail больше допустимой!!!<BR>";
   }
   if (length($pass) > 16 || length($pass) < 6) {
      $err_add .= "Длина пароля должна быть в пределах ";
      $err_add .= "от 6 до 16 символов.<BR>";
   }
   if ($pass !~ /^[a-z0-9_\-]{6,16}$/i) {
      $err_add .= "В пароле допустимы только буквы ";
      $err_add .= "A-Z (a-z) или цифры 0-9<BR>";
   }
   if ($err_add eq "") { # Если ошибок нет
      my $q = "SELECT * FROM user WHERE email='$emails'";
      my $res_add = $db->prepare($q);
      $res_add->execute();
      if (!$res_add->err) {
         if ($res_add->rows() == 0) {
            $pass = md5_hex($pass);
            my $query_add = "INSERT INTO user VALUES ";
            $query_add .= "(NULL, '$emails', '$pass', 'y')";
            my $result = $db->do($query_add);
```

```
            if ($result == 1) {
               my $sess = CGI::Session->new("driver:file", undef,
                                {Directory=>$DATA{'TMP_PATH'}})
                                or die CGI::Session->errstr();
               $sess->name('SID');
               my $cookies = cookie(-name=>$sess->name(),
                                    -value=>$sess->id(), -path=>'/');
               print "Set-Cookie: $cookies\n";
               $sess->param(-name=>'Login', -value=>$emails);
               $sess->param(-name=>'Password', -value=>$pass);
               $sess->expire($DATA{'TIME_SESS'});
               $sess->flush();
               print "Location: add.pl\n\n";
               exit();
            }
            else {
               $err_add .= "Произошла ошибка при регистрации<BR>";
               $err_add .= "Попробуйте сделать запрос ";
               $err_add .= "через некоторое время<BR>";
            }
         }
         else {
            $err_add .= "E-mail $emails уже зарегистрирован ";
            $err_add .= "ранее!!!<BR>";
            $err_add .= "Восстановить пароль можно, заполнив форму ";
            $err_add .= "ниже<BR>";
            $emails = "";
            $pass = "";
         }
      }
      else {
         $err_add .= "Произошла ошибка при регистрации<BR>";
         $err_add .= "Попробуйте сделать запрос ";
         $err_add .= "через некоторое время<BR>";
      }
   }
}
```

```
# Восстанавливаем пароль
if (defined(param('mail_pass'))) {
   my $mails = param('mails') || "";
   my $pattern = qr/^[a-z0-9_\.\-]+\@([a-z0-9\-]+\.)+[a-z]{2,4}$/;
   if ($mails !~ /$pattern/i) {
      $err_mail = "Недопустимый адрес E-mail !!!<BR>";
   }
   if (length($mails) > 50) {
      $err_mail .= "Длина E-mail больше допустимой!!!<BR>";
   }
   if ($err_mail eq "") { # Если ошибок нет
      my $query_mail = "SELECT * FROM user WHERE email='$mails'";
      my $res_mail = $db->prepare($query_mail);
      $res_mail->execute();
      if (!$res_mail->err) {
         if ($res_mail->rows() == 1) {
            my @row = $res_mail->fetchrow_array();
            my $id_user = $row[0];
            # Генерируем новый пароль
            my $new_passw = &f_passw_generator();
            my $pass2 = md5_hex($new_passw);
            my $query = "UPDATE user SET passw='$pass2' ";
            $query .= "WHERE id_user='$id_user' LIMIT 1";
            my $result = $db->do($query);
            if ($result == 1) {
               my $header = "From: " . $DATA{'MAIL_POST'} . "\n";
               $header .= "To: $mails\n";
               $header .= "Subject: Регистрационные данные\n";
               $header .= "Content-Type: text/plain; ";
               $header .= "charset=windows-1251\n";
               $header .= "Content-Transfer-Encoding: 8bit\n\n";
               $msg = "Добрый день!\n\n";
               $msg .= "Пароль для доступа $new_passw \n\n";
               $msg .= $DATA{'URL_SITE'} . "\n";
               open(MAIL, "| /usr/sbin/sendmail -t") or die("Ошибка $!");
               print MAIL $header;
               print MAIL $msg;
```

```
            close(MAIL) or die("Ошибка $!");
            $msg = "Пароль отправлен на E-mail $mails";
         }
         else {
            $err_mail .= "Произошла ошибка.<BR>";
            $err_mail .= "Попробуйте сделать запрос ";
            $err_mail .= "через некоторое время<BR>";
         }
      }
      else {
         $err_mail .= "E-mail не найден<BR>";
      }
   }
   else {
      $err_mail .= "Произошла ошибка.<BR>";
      $err_mail .= "Попробуйте сделать запрос ";
      $err_mail .= "через некоторое время<BR>";
   }
}

print "Content-type: text/html; charset=windows-1251\n\n";
# Заголовок
my $title = "Вход в Личный кабинет";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);

print <<SCRIPT_COD;
<SCRIPT language="JavaScript">
<!--
function f_test_mail(m_Mail) {
   c_Reg = /^[a-z0-9_\\.\\-]+\@([a-z0-9\\-]+\\.)+[a-z]{2,4}\$/i;
   if (!c_Reg.test(m_Mail)) return false;
   return true;
}
```

```
function f_test_passw(m_Pass) {
   c_Reg = /^[a-z0-9_\\.\\-]{6,16}\$/i;
   if (!c_Reg.test(m_Pass)) return false;
   return true;
}
function f_submit_enter() {
   var m_f = document.form_enter;
   if (m_f.login.value=="" || m_f.passw.value=="") {
      window.alert("Не заполнено обязательное поле");
      m_f.login.focus();
      return false;
   }
   if (m_f.login.value.length>50 || !f_test_mail(m_f.login.value)) {
      window.alert("Недопустимое значение поля E-mail");
      m_f.login.focus();
      return false;
   }
   if (!f_test_passw(m_f.passw.value)) {
      m_msg = "Недопустимое значение поля Пароль\\n";
      m_msg += "Допустимо от 6 до 16 символов\\n";
      m_msg += "Русские буквы использовать нельзя";
      window.alert(m_msg);
      m_f.passw.focus();
      return false;
   }
return true;
}
function f_submit_add() {
   var m_f = document.form_add;
   if (m_f.emails.value=="" || m_f.pass.value=="") {
      window.alert("Не заполнено обязательное поле");
      m_f.emails.focus();
      return false;
   }
   if (m_f.emails.value.length>50 || !f_test_mail(m_f.emails.value)) {
      window.alert("Недопустимое значение поля E-mail");
      m_f.emails.focus();
```

```
      return false;
   }
   if (!f_test_passw(m_f.pass.value)) {
      m_msg = "Недопустимое значение поля Пароль\\n";
      m_msg += "Допустимо от 6 до 16 символов\\n";
      m_msg += "Русские буквы использовать нельзя";
      window.alert(m_msg);
      m_f.pass.focus();
      return false;
   }
return true;
}
function f_submit_mail() {
   var m_f = document.form_mail;
   if (m_f.mails.value=="") {
      window.alert("Поле не заполнено");
      m_f.mails.focus();
      return false;
   }
   if (m_f.mails.value.length>50 || !f_test_mail(m_f.mails.value)) {
      window.alert("Недопустимое значение поля E-mail");
      m_f.mails.focus();
      return false;
   }
return true;
}
//-->
</SCRIPT>
SCRIPT_COD

&f_table_2_start(); # Выводим таблицу второго уровня
print "<H1>Личный кабинет</H1><BR>\n";
if ($err_enter ne "") {
   print "<DIV class=\"err\">$err_enter</DIV><BR>\n";
}
print <<FORM_ENTER;
<TABLE width="100%" border="0" cellpadding="1" align="center">
<FORM method="POST" name="form_enter" id="form_enter"
```

```
onsubmit="return f_submit_enter();">
<TR><TD align="right" width="50%">
<SPAN class="bold">E-mail: </SPAN></TD><TD>
<INPUT type="text" name="login" id="login" maxlength="50" size="23"
class="txt_frm" value="$login">
</TD></TR><TR><TD align="right" width="50%">
<SPAN class="bold">Пароль: </SPAN></TD><TD>
<INPUT type="password" name="passw" id="passw" maxlength="16"
size="23" class="txt_frm">
</TD></TR><TR><TD align="right" width="50%">.:</TD><TD>
<INPUT type="submit" name="enter" value="Вход" class="txt_frm">
</TD></TR>
</FORM>
</TABLE><BR>
FORM_ENTER
if ($err_add ne "") {
   print "<DIV class=\"err\">$err_add</DIV><BR>\n";
}
print <<FORM_ADD;
<TABLE width="100%" border="0" cellpadding="1" align="center">
<FORM method="POST" name="form_add" id="form_add"
onsubmit="return f_submit_add();">
<TR><TD align="right" width="50%">
<SPAN class="bold">E-mail: </SPAN></TD><TD>
<INPUT type="text" name="emails" id="emails" maxlength="50" size="23"
class="txt_frm" value="$emails">
</TD></TR><TR><TD align="right" width="50%">
<SPAN class="bold">Пароль: </SPAN></TD><TD>
<INPUT type="password" name="pass" id="pass" maxlength="16" size="23"
class="txt_frm" value="$pass">
</TD></TR>
<TR><TD align="right" width="50%">.:</TD><TD>
<INPUT type="submit" name="add" value="Регистрация" class="txt_frm">
</TD></TR>
</FORM>
</TABLE><BR>
FORM_ADD
```

```
if ($err_mail ne "") {
   print "<DIV class=\"err\">$err_mail</DIV><BR>\n";
}
if ($msg ne "") {
   print "<DIV class=\"ok\">$msg</DIV><BR>\n";
}
print <<FORM_MAIL;
<TABLE width="100%" border="0" cellpadding="1" align="center">
<FORM method="POST" name="form_mail" id="form_mail"
onsubmit="return f_submit_mail();">
<TR><TD align="right" width="50%">
<SPAN class="bold">E-mail: </SPAN></TD><TD>
<INPUT type="text" name="mails" id="mails" maxlength="50" size="23"
class="txt_frm">
</TD></TR><TR><TD align="right" width="50%">.:</TD><TD>
<INPUT type="submit" name="mail_pass" value="Восстановить пароль"
class="txt_frm">
</TD></TR>
</FORM>
</TABLE><BR>
FORM_MAIL

&f_table_2_end(); # Конец таблицы второго уровня
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();
```

Почему мы генерируем новый пароль, а не просто высылаем сохраненный в базе данных? Все дело в том, что при сохранении в базе данных мы шифруем пароль с помощью функции `md5_hex()`. Способа расшифровать данный пароль нет. Поэтому для восстановления необходимо генерировать новый пароль. Для сравнения введенного пароля с сохраненным в базе достаточно зашифровать введенный пароль, а затем сравнить.

В ряде случаев для восстановления пароля можно использовать другой метод. При запросе на восстановление высылать на E-mail не новый пароль, а лишь ссылку с уникальным идентификатором. А уже после перехода по ссылке генерировать новый пароль.

5.14.2. Добавление новых сайтов в базу

Алгоритм работы программы следующий:

1. Запускаем сессию.
2. Проверяем, не истекло ли время сессии и совпадают ли IP-адреса. Если это не так, то удаляем сессию с помощью функции `f_delete_session()`, определенной в конце программы.
3. С помощью метода `param()` получаем значения переменных сессии и сохраняем их в переменных `$sess_login` (логин) и `$sess_pass` (пароль).
4. Если переменные не определены, то удаляем сессию.
5. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем выполнение скрипта.
6. Делаем запрос для определения идентификатора пользователя, а затем сохраняем его в переменной `$id_user`.
7. Продлеваем время сессии.
8. Если форма отправлена (параметр `param('go')` будет существовать):
   - с помощью функции `param()` из модуля `CGI` получаем параметры, переданные скрипту, и сохраняем их в переменных `$urls` (URL-адрес сайта), `$rubr` (идентификатор рубрики), `$titles` (название ресурса) и `$deskr` (описание ресурса);
   - заменяем все теги на HTML-эквиваленты и экранируем специальные символы;
   - проверяем значения переменных на допустимость;
   - с помощью метода `host()` из модуля `URI` вычленяем адрес хоста из URL-адреса;
   - если ошибок нет, то проверяем, не зарегистрирован ли сайт ранее;
   - если сайт не был зарегистрирован, то добавляем его в базу данных.
9. Если существуют ошибки, то выводим их описание.
10. Если сайт успешно добавлен в базу, то выводим подтверждение.
11. Выводим первую часть формы.
12. Заполняем список рубрик из базы данных.
13. Выводим вторую часть формы.
14. Размещаем ссылку для выхода из системы.
15. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.40.

Листинг 5.40. Содержимое файла /cgi-bin/user/add.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
use CGI::Session;
use URI;
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :user &f_date_gm );
my $SID;
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
}
else {
   print "Location: index.pl\n\n";
   exit();
}
my $sess = CGI::Session->new("driver:file", $SID,
                  {Directory=>$DATA{'TMP_PATH'}})
                  or die CGI::Session->errstr();
$sess->name('SID');
&f_delete_session() if ($sess->is_empty());
# Если время сессии истекло
&f_delete_session() if ($sess->is_expired());
# Если IP-адреса не совпадают
&f_delete_session() if ($sess->remote_addr() ne $ENV{'REMOTE_ADDR'});
my $sess_login = $sess->param('Login');
my $sess_pass = $sess->param('Password');
```

```
unless (defined($sess_login) && defined($sess_pass)) {
   &f_delete_session();
}

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "Content-type: text/html; charset=windows-1251\n\n";
   print "<DIV style=\"font-size: 16px; font-weight: bold; ";
   print "text-align: center; color: #FF0000\">";
   print "Не удалось установить соединение с базой данных</DIV>";
   exit();
}
$db->do("SET NAMES cp1251");

my $id_user;
$sess_login = $db->quote($sess_login);
$sess_pass = $db->quote($sess_pass);
my $query_enter = "SELECT * FROM user ";
$query_enter .= "WHERE email=$sess_login AND passw=$sess_pass";
my $res = $db->prepare($query_enter);
$res->execute();
if (!$res->err) {
   if ($res->rows() == 1) {
      my @row = $res->fetchrow_array();
      $id_user = $row[0];
   }
   else {
      &f_delete_session();
   }
}
else {
   print "Content-type: text/html; charset=windows-1251\n\n";
   print "<DIV style=\"font-size: 16px; font-weight: bold; ";
   print "text-align: center; color: #FF0000\">";
   print "Ошибка!</DIV>";
```

```
   exit();
}
$res->finish();
# Продлеваем время сессии
$sess->atime(time());
$sess->flush();

my $err_add = "";
my $ok_add = "";
my $titles = "";
my $deskr = "";
my $urls = "";
my $rubr = 0;

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";
# Заголовок
my $title = "Добавить сайт";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

print "<H1>Личный кабинет</H1><BR>\n";
print <<SCRIPT_COD;
<SCRIPT language="JavaScript">
<!--
function f_submit() {
   var m_frm=document.frm;
   if (m_frm.urls.value.length<8 || m_frm.titles.value=="" ||
      m_frm.deskr.value=="") {
      window.alert("Поле не заполнено");
```

```
      return false;
   }
   if (m_frm.titles.value.length>70 || m_frm.deskr.value.length>500) {
      var m_msg = "Длина Названия ресурса не должна превышать ";
      m_msg += "70 символов, а Описания — 500";
      window.alert(m_msg);
      return false;
   }
   c_Reg = /^http:\\/\\/(www\\.)?([a-z0-9\\-]+\\.)+[a-z]{2,4}\$/i;
   if (!c_Reg.test(m_frm.urls.value)) {
      m_msg = "Недопустимый URL\\n";
      m_msg += "Правильно — \\"http://www.mail.ru\\"\\n";
      m_msg += "Неправильно — \\"http://www.mail.ru/\\"";
      window.alert(m_msg);
      m_frm.urls.focus();
      return false;
   }
   if (m_frm.rubr.options[m_frm.rubr.selectedIndex].value==0) {
      window.alert("Рубрика не выбрана");
      return false;
   }
return true;
}
//-->
</SCRIPT>
SCRIPT_COD

if (defined(param('go'))) { # Если форма отправлена
   $urls = param('urls') || "";
   $rubr = param('rubr') || 0;
   $titles = param('titles') || "";
   $deskr = param('deskr') || "";
   # Заменяем спецсимволы
   $titles =~ s/&/&amp;/g;
   $titles =~ s/</&lt;/g;
   $titles =~ s/>/&gt;/g;
   $titles =~ s/"/&quot;/g;
```

```
$titles =~ s/\r//g;
$titles =~ s/\t/ /g;
$titles =~ s/\\/\\\\/g;
$titles =~ s/'/\\'/g;
$deskr =~ s/&/&amp;/g;
$deskr =~ s/</&lt;/g;
$deskr =~ s/>/&gt;/g;
$deskr =~ s/"/&quot;/g;
$deskr =~ s/\r//g;
$deskr =~ s/\t/ /g;
$deskr =~ s/\\/\\\\/g;
$deskr =~ s/'/\\'/g;
# Удаляем символы новой строки
{
   local $/ = "";
   chomp($titles);
   chomp($deskr);
}
$titles =~ s/\n/ /g;
$deskr =~ s/\n/ /g;
if (length($titles) > 70 || length($deskr) > 500) {
   $err_add .= "Длина Названия ресурса не должна превышать ";
   $err_add .= "70 символов, а Описания — 500<BR>";
}
 if ($titles =~ /[a-za-я0-9,\.\"\']{26}/i) {
   $err_add .= "Поле Название ресурса содержит более 25 ";
   $err_add .= "символов без пробела!<BR>\n";
 }
 if ($deskr =~ /[a-za-я0-9,\.\"\']{26}/i) {
   $err_add .= "Поле Описание содержит более 25 символов ";
   $err_add .= "без пробела!<BR>\n";
 }
if (length($titles) < 2 || length($deskr) < 2) {
   $err_add .= "Не заполнено обязательное поле<BR>";
}
my $host;
if ($urls !~ /^http:\/\/(www\.)?([a-z0-9\-]+\.)+[a-z]{2,4}$/i ||
    length($urls)>200)
```

```
{
   $err_add .= "Недопустимый URL !<BR>";
}
else {
   $urls = lc($urls);
   my $url = URI->new($urls);
   $host = $url->host();
   if ($host =~ /^www\./) {
      $host = substr($host, 4);
   }
}
if ($rubr !~ /^[0-9]+$/ || $rubr == 0) {
   $err_add .= "Рубрика не выбрана или имеет неправильный формат<BR>";
}
if ($err_add eq "") { # Если ошибок нет
   my $query_test = "SELECT * FROM site WHERE url_site_dop='$host'";
   my $res_test = $db->prepare($query_test);
   $res_test->execute();
   if (!$res_test->err) {
      if ($res_test->rows() == 0) {
         # Если сайт не зарегистрирован ранее, то добавляем его
         my $q = "INSERT INTO site VALUES (NULL, '$rubr', ";
         $q .= "'$id_user', '$urls', '$host', '$titles', '$deskr', ";
         $q .= "'n', 0, CURDATE())";
         my $res_q = $db->do($q);
         if ($res_q == 1) {
            $ok_add="Сайт $urls добавлен в базу данных<BR>";
            $urls = "http://";
            $rubr = 0;
            $titles = $deskr = "";
         }
         else {
            $err_add .= "Попробуйте сделать запрос через некоторое ";
            $err_add .= "время<BR>";
         }
      }
```

```
         else {
            $err_add .= "Сайт $urls был зарегистрирован ранее<BR>";
            $urls = "http://";
            $rubr = 0;
            $titles = $deskr = "";
         }
      }
      else {
         $err_add .= "Ошибка при добавлении сайта<BR>";
      }
      $res_test->finish();
   }
}
else {
   $urls = "http://";
}

if ($err_add ne "") {
   print "<DIV class=\"err\">$err_add</DIV><BR>\n";
   $titles =~ s/\\\\/\\/g;
   $titles =~ s/\\'/'/g;
   $deskr =~ s/\\\\/\\/g;
   $deskr =~ s/\\'/'/g;
}
if ($ok_add ne "") {
   print "<DIV class=\"ok\">$ok_add</DIV><BR>\n";
}
print <<FORM_START;
<FORM method="POST" name="frm" id="frm" onsubmit="return f_submit();">
<TABLE width="100%" border="0" cellpadding="1" align="center">
<TR><TD align="right" width="40%">
<SPAN class="bold">URL: </SPAN>
</TD><TD>
<INPUT type="text" name="urls" id="urls" size="40" value="$urls"
class="txt_frm">
</TD></TR>
</TABLE><BR>
```

```
<DIV align="center">
<SELECT name="rubr" id="rubr" class="select_frm">
<OPTION value="0">-----------Выберите рубрику---</OPTION>
FORM_START

my $query_rubr = "SELECT * FROM rubr ORDER BY name_rubr";
my $res_rubr = $db->prepare($query_rubr);
$res_rubr->execute();
if (!$res_rubr->err) {
   while (my @row_rubr = $res_rubr->fetchrow_array()) {
      print "<OPTION value=\"$row_rubr[0]\"";
      if ($rubr == $row_rubr[0]) {
         print " selected>";
      }
      else {
         print ">";
      }
      $row_rubr[1] =~ s/"/&quot;/g;
      print $row_rubr[1];
      print "</OPTION>\n";
   }
}
$res_rubr->finish();

print <<FORM_END;
</SELECT>
</DIV><BR>
<TABLE width="100%" border="0" cellpadding="1" align="center">
<TR><TD align="right" width="40%">
<SPAN class="bold">Название ресурса: </SPAN>
</TD><TD>
<INPUT type="text" name="titles" id="titles" size="40" maxlength="70"
class="txt_frm" value="$titles">
</TD></TR>
<TR><TD align="right" width="40%">
<SPAN class="bold">Описание ресурса:<BR> (до 500 символов)</SPAN>
</TD><TD>
```

```
<TEXTAREA name="deskr" id="deskr" class="textarea_frm"
style="width: 300px; height: 150px">$deskr</TEXTAREA>
</TD></TR>
<TR><TD align="right" width="40%">.:</TD><TD>
<INPUT type="submit" name="go" value="Добавить сайт" class="txt_frm">
</TD></TR>
</TABLE>
</FORM><BR>
FORM_END
print "<DIV align=\"center\">";
print "<A href=\"exit.pl\">Выйти из системы</A></DIV>\n";
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_user();

# Удаление сессии
sub f_delete_session {
   my $cookies = cookie(-name=>$sess->name(), -value=>'',
      -expires=>'-1d', -path=>'/', -domain=>'.perlbook.ru');
   print "Set-Cookie: $cookies\n";
   $sess->delete();
   $sess->flush();
   print "Location: index.pl\n\n";
   exit();
}
```

Для выхода из системы создадим файл exit.pl с содержимым листинга 5.41.

Листинг 5.41. Содержимое файла /cgi-bin/user/exit.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use CGI qw( :standard);
use CGI::Session;
```

```
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript;
my $SID;
if (defined(cookie('SID'))) {
   $SID = cookie('SID');
}
else {
   print "Location: index.pl\n\n";
   exit();
}
my $sess = CGI::Session->new("driver:file", $SID,
                  {Directory=>$DATA{'TMP_PATH'}})
                  or die CGI::Session->errstr();
$sess->name('SID');
my $cookies = cookie(-name=>$sess->name(), -value=>'', -expires=>'-1d',
                        -path=>'/');
print "Set-Cookie: $cookies\n";
$sess->delete();
$sess->flush();
print "Location: index.pl\n\n";
exit();
```

5.15. Администраторская часть сайта

В качестве примера создадим Личный кабинет администратора сайта и защитим его не с помощью механизма сессий, а посредством сервера Apache. Для этого в каталоге /cgi-bin/admin/ необходимо разместить файл .htaccess со следующим содержимым:

```
AuthType Basic
AuthName "Enter password"
AuthUserFile <Путь к файлу .htpasswd>
<Limit GET POST>
   require valid-user
</Limit>
```

Кроме того, необходимо создать файл .htpasswd, в котором должен быть пароль для доступа. Пароль создается с помощью программы htpasswd.exe следующим образом:

```
htpasswd -c <Путь к файлу .htpasswd> <Пользователь>
```

После ввода команды необходимо будет ввести пароль, а затем его повторить. Для добавления нового пользователя в уже существующий файл используется команда:

```
htpasswd -b <Путь к файлу .htpasswd> <Пользователь> <Пароль>
```

Обратите внимание, что вместо флага `-c` мы использовали флаг `-b`, а также указали пароль сразу после имени пользователя. Если использовать флаг `-c`, то файл будет перезаписан, и соответственно вся информация будет удалена.

К сожалению, программа htpasswd.exe не входит в базовый пакет Денвера. Как создать файл .htpasswd на виртуальном хостинге, мы рассмотрим в *разд. 6.11*. Сейчас, просто ради примера, разместим в каталоге /cgi-bin/admin/ файл .htaccess со следующими директивами:

```
AuthType Basic
AuthName "Enter password"
AuthUserFile /home/site.ru/cgi-bin/admin/.htpasswd
<Limit GET POST>
   require valid-user
</Limit>
```

Теперь разместим в каталоге /cgi-bin/admin/ файл .htpasswd со следующим содержимым:

```
user1:$apr1$Wr......$DKAb1szOCk1wPuErprLfe1
```

При попытке зайти в Личный кабинет администратора будет выведено окно, в которое необходимо ввести логин (`user1`) и пароль (`pass1`).

В каталоге /cgi-bin/admin/ создадим следующие файлы:

- index.pl — вывод статистики и сообщений об ошибках;
- rubr.pl — для добавления новой рубрики, удаления или переименования существующей;
- moder.pl — для вывода сайтов, находящихся на модерации и результатов поиска по URL-адресу с возможностью одобрения или удаления выделенных сайтов;
- catalog.pl — для редактирования описания сайта;
- gbook.pl — вывод содержимого гостевой книги с выделением новых сообщений, а также возможностью одобрения или удаления выделенных сообщений.

5.15.1. Добавление, изменение и удаление рубрик

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. Если форма для добавления новой рубрики отправлена (параметр `param('rubr_add')` будет существовать):
   - с помощью метода `param()` получаем переданное название новой рубрики и сохраняем название в переменной `$rubrika`;
   - проверяем корректность названия новой рубрики;
   - добавляем защитные слэши перед спецсимволами с помощью метода `quote()`;
   - проверяем название новой рубрики на уникальность;
   - добавляем рубрику в базу данных.
3. Если форма для удаления рубрики отправлена (параметр `param('del_rubr')` будет существовать):
   - с помощью метода `param()` получаем переданный идентификатор рубрики и сохраняем его в переменной `$rubrikator`;
   - проверяем корректность введенных данных;
   - проверяем существование сайтов, зарегистрированных в данной рубрике;
   - если зарегистрированных сайтов нет, то удаляем рубрику.
4. Если форма для изменения названия рубрики отправлена (параметр `param('change')` будет существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в переменных `$rubrikator` (идентификатор рубрики) и `$change_rubr` (новое название рубрики);
   - проверяем корректность введенных данных;
   - добавляем защитные слэши перед спецсимволами с помощью метода `quote()`;
   - проверяем название новой рубрики на уникальность;
   - проверяем существование идентификатора рубрики;
   - изменяем название рубрики.

5. Если возникли ошибки при добавлении новой рубрики, то выводим их описание.
6. В случае успешного добавления новой рубрики выводим подтверждение.
7. Выводим форму для добавления новой рубрики.
8. Если возникли ошибки при изменении или удалении рубрики, то выводим их описание.
9. В случае успешного изменения или удаления рубрики выводим подтверждение.
10. Выводим форму для изменения или удаления рубрики.
11. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.42.

Листинг 5.42. Содержимое файла /cgi-bin/admin/rubr.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Администрирование";
# Описание страницы
my $description = "";
```

```
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_admin();
   exit();
}
$db->do("SET NAMES cp1251");

my $err_rubr = "";
my $ok_rubr = "";
my $err_rubrikator = "";
my $ok_rubrikator = "";

# Добавление новой рубрики
if (defined(param('rubr_add'))) {
   my $rubrika = param('rubrika') || "";
   if (length($rubrika) > 150) {
      $err_rubr .= "Длина Рубрики больше допустимой!!!<BR>";
   }
   if (length($rubrika) == 0) {
      $err_rubr .= "Поле не заполнено!!<BR>";
   }
   if ($err_rubr eq "") {
      $rubrika = $db->quote($rubrika);
      my $q = "SELECT * FROM rubr WHERE name_rubr=$rubrika";
      my $res_rubr = $db->prepare($q);
      $res_rubr->execute();
      if (!$res_rubr->err) {
         if ($res_rubr->rows() == 0) {
            my $query_rubr = "INSERT INTO rubr VALUES (NULL, $rubrika)";
```

```
            my $res = $db->do($query_rubr);
            if ($res == 1) {
               $ok_rubr = "Рубрика успешно добавлена<BR>";
            }
            else {
               $err_rubr .= "Попробуйте сделать запрос через некоторое ";
               $err_rubr .= "время<BR>";
            }
         }
         else {
            $err_rubr .= "Рубрика была зарегистрирована ранее!!!<BR>";
         }
      }
      else {
         $err_rubr .= "Ошибка.<BR>";
      }
      $res_rubr->finish();
   }
}
# Удаление рубрики
if (defined(param('del_rubr'))) {
   my $rubrikator = param('rubrikator') || 0;
   if ($rubrikator !~ /^[0-9]+$/ || $rubrikator == 0) {
      $err_rubrikator .= "Недопустимый формат номера рубрики!!!<BR>";
   }
   else {
      my $q_test = "SELECT * FROM rubr WHERE id_rubr=$rubrikator";
      my $res_test = $db->prepare($q_test);
      $res_test->execute();
      if (!$res_test->err) {
         if ($res_test->rows() != 1) {
            $err_rubrikator .= "Рубрика не найдена<BR>";
         }
      }
      else {
         $err_rubrikator .= "Ошибка.<BR>";
      }
```

```
      $res_test->finish();
   }
   if ($err_rubrikator eq "") {
      my $q = "SELECT * FROM site WHERE id_rubr=$rubrikator";
      my $res_rubr = $db->prepare($q);
      $res_rubr->execute();
      if (!$res_rubr->err) {
         if ($res_rubr->rows() == 0) {
            my $query_rubrikator = "DELETE FROM rubr WHERE ";
            $query_rubrikator .= "id_rubr=$rubrikator LIMIT 1";
            my $res = $db->do($query_rubrikator);
            if ($res == 1) {
               $ok_rubrikator = "Рубрика успешно удалена<BR>";
            }
            else {
               $err_rubrikator .= "Попробуйте сделать запрос через ";
               $err_rubrikator .= "некоторое время<BR>";
            }
         }
         else {
            $err_rubrikator .= "Нельзя удалить рубрику, т. к. есть ";
            $err_rubrikator .= "сайты с указанной рубрикой!!!<BR>";
         }
      }
      else {
         $err_rubrikator .= "Ошибка.<BR>";
      }
      $res_rubr->finish();
   }
}
# Изменение названия рубрики
if (defined(param('change'))) {
   my $rubrikator = param('rubrikator') || 0;
   my $change_rubr = param('change_rubr') || "";
   if ($rubrikator !~ /^[0-9]+$/ || $rubrikator == 0) {
      $err_rubrikator .= "Недопустимый формат номера рубрики!!!<BR>";
   }
```

```
if (length($change_rubr) > 150) {
   $err_rubrikator .= "Длина Рубрики больше допустимой!!!<BR>";
}
if (length($change_rubr) == 0) {
   $err_rubrikator .= "Поле не заполнено!!!<BR>";
}
if ($err_rubrikator eq "") {
   $change_rubr = $db->quote($change_rubr);
   my $q_change_rubr = "SELECT * FROM rubr ";
   $q_change_rubr .= "WHERE name_rubr=$change_rubr";
   my $res_change_rubr = $db->prepare($q_change_rubr);
   $res_change_rubr->execute();
   if (!$res_change_rubr->err) {
      if ($res_change_rubr->rows() != 0) {
         $err_rubrikator .= "Рубрика с таким названием уже ";
         $err_rubrikator .= "существует!!!<BR>";
      }
   }
   else {
      $err_rubrikator .= "Ошибка.<BR>";
   }
   $res_change_rubr->finish();
   if ($err_rubrikator eq "") {
      my $q = "SELECT * FROM rubr WHERE id_rubr=$rubrikator";
      my $res_change_r = $db->prepare($q);
      $res_change_r->execute();
      if (!$res_change_r->err) {
         if ($res_change_r->rows() == 1) {
            my $query_change_rubr = "UPDATE rubr SET ";
            $query_change_rubr .= "name_rubr=$change_rubr ";
            $query_change_rubr .= "WHERE id_rubr=$rubrikator LIMIT 1";
            my $res = $db->do($query_change_rubr);
            if ($res == 1) {
               $ok_rubrikator = "Рубрика успешно переименована<BR>";
            }
            else {
               $err_rubrikator .= "Попробуйте сделать запрос через ";
```

```
                $err_rubrikator .= "некоторое время<BR>";
            }
         }
         else {
            $err_rubrikator .= "Рубрика не найдена<BR>";
         }
      }
      else {
         $err_rubrikator .= "Ошибка.<BR>";
      }
      $res_change_r->finish();
   }
  }
}

print <<SCRIPT_COD;
<SCRIPT language="JavaScript">
<!--
function f_submit_rubr() {
   var m_f = document.form_rubr;
   if (m_f.rubrika.value=="") {
      window.alert("Поле не заполнено");
      m_f.rubrika.focus();
      return false;
   }
   if (m_f.rubrika.value.length>150) {
      window.alert("Недопустимое значение поля Рубрика");
      m_f.rubrika.focus();
      return false;
   }
return true;
}

function f_change() {
   var m_frm = document.frm_rubrikator;
   m_frm.change_rubr.value =
   m_frm.rubrikator.options[m_frm.rubrikator.selectedIndex].text;
}
```

```
function f_change_rubr() {
   var m_frm = document.frm_rubrikator;
   if (m_frm.change_rubr.value=="") {
      window.alert("Поле не заполнено");
      m_frm.change_rubr.focus();
      event.returnValue = false;
   }
   else {
      var m_msg = "Вы действительно хотите переименовать\\nрубрику \\"";
      var m_rbk = m_frm.rubrikator;
      m_msg += m_rbk.options[m_rbk.selectedIndex].text;
      m_msg += "\\" в \\"" + m_frm.change_rubr.value + "\\"?";
      if (window.confirm(m_msg)) {}
      else event.returnValue = false;
   }
}
function f_del_rubr() {
   var m_frm= document.frm_rubrikator;
   var m_msg= "Вы действительно хотите УДАЛИТЬ\\nрубрику \\"";
   var m_rbk = m_frm.rubrikator;
   m_msg += m_rbk.options[m_rbk.selectedIndex].text;
   m_msg += "\\"?";
   if (window.confirm(m_msg)) {}
   else event.returnValue = false;
}
//-->
</SCRIPT>
SCRIPT_COD
print "<H1>Рубрикатор</H1><BR><BR>\n";
if ($err_rubr ne "") {
   print "<DIV class=\"err\">$err_rubr</DIV><BR>\n";
}
if ($ok_rubr ne "") {
   print "<DIV class=\"ok\">$ok_rubr</DIV><BR>\n";
}
print <<FORM_COD;
<TABLE width="450" border="0" cellpadding="1" align="center">
<FORM method="POST" name="form_rubr" id="form_rubr"
```

```
onsubmit="return f_submit_rubr();">
<TR><TD align="right" width="50%">
<SPAN class="bold">Рубрика: </SPAN></TD><TD>
<INPUT type="text" name="rubrika" id="rubrika" maxlength="50"
size="23" class="txt_frm">
</TD></TR><TR><TD align="right" width="50%">.:</TD><TD>
<INPUT type="submit" name="rubr_add" value="Добавить рубрику"
class="txt_frm">
</TD></TR>
</FORM>
</TABLE><BR>
FORM_COD
if ($err_rubrikator ne "") {
   print "<DIV class=\"err\">$err_rubrikator</DIV><BR>\n";
}
if ($ok_rubrikator ne "") {
   print "<DIV class=\"ok\">$ok_rubrikator</DIV><BR>\n";
}
print <<FORM_START;
<TABLE width="450" border="0" cellpadding="1" align="center">
<FORM name="frm_rubrikator" id="frm_rubrikator" method="POST">
<TR><TD align="right" width="50%">
<SPAN class="bold">Рубрика: </SPAN></TD><TD>
<SELECT name="rubrikator" id="rubrikator" class="select_frm"
style="width: 154px" onchange="f_change();">
FORM_START

my $query_rubr = "SELECT * FROM rubr ORDER BY name_rubr";
my $res_rubr = $db->prepare($query_rubr);
$res_rubr->execute();
if (!$res_rubr->err) {
   while (my @row_rubr = $res_rubr->fetchrow_array()) {
      print "<OPTION value=\"$row_rubr[0]\">";
      $row_rubr[1] =~ s/"/&quot;/g;
      print $row_rubr[1];
      print "</OPTION>\n";
   }
}
$res_rubr->finish();
```

```
print <<FORM_END;
</SELECT>
</TD></TR><TR><TD align="right" width="50%">
<SPAN class="bold">Изменить на: </SPAN></TD><TD>
<INPUT type="text" name="change_rubr" id="change_rubr"
maxlength="50" size="23" class="txt_frm">
</TD></TR><TR><TD align="right" width="50%">
<INPUT type="submit" name="del_rubr" value="Удалить рубрику"
class="txt_frm"
style="background-color: #FF7F50" onclick="f_del_rubr();">
</TD><TD>
<INPUT type="submit" name="change" value="Изменить название"
class="txt_frm" onclick="f_change_rubr();">
</TD></TR>
</FORM>
</TABLE><BR>
FORM_END

# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_admin();
```

Обратите внимание, что форма для изменения или удаления рубрики содержит две кнопки **Submit**. При нажатии на одну из них будут существовать параметры `param('change')` или `param('del_rubr')` соответственно. Проверка на существование параметра позволяет обработать нажатие соответствующей кнопки.

Если в таблице `site` есть сайты, зарегистрированные на удаляемую рубрику, то рубрику удалять нельзя, т. к. это повредит целостность базы данных. Для предотвращения случайного нажатия кнопки **Удалить** мы с помощью языка JavaScript выводим диалоговое окно подтверждения.

5.15.2. Вывод сайтов, требующих проверки

Все добавляемые сайты должны пройти проверку модератором. Для этого при добавлении сайта мы указываем статус "на модерации". Сайты, имеющие такой статус, в каталоге не выводятся.

Для вывода сайтов, требующих проверки, создадим файл moder.pl. Кроме того, этот файл будет использоваться для отображения результатов поиска по URL-адресу.

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. Если форма для удаления сайта отправлена (параметры `param('del_msg')` и `param('check')` будут существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в массиве `@ch`;
   - в цикле проверяем идентификатор сайта, и если это число, то удаляем сайт.
3. Если форма для одобрения сайта отправлена (параметры `param('upd_msg')` и `param('check')` будут существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в массиве `@ch`;
   - в цикле проверяем идентификатор сайта, и если это число, то изменяем статус сайта на "активен".
4. С помощью метода `param()` получаем переданный методом `GET` номер страницы и сохраняем его в переменной `$page`.
5. Если форма для поиска по URL-адресу отправлена (параметр `param('search')` будет существовать):
   - с помощью метода `param()` получаем переданный поисковый запрос и сохраняем его в переменной `$text`;
   - удаляем все специальные символы;
   - проверяем значение переменной на допустимость;
   - добавляем защитные слэши перед спецсимволами;
   - создаем поисковый запрос.
6. Если поисковый запрос не существует (поисковая форма не отправлена), создаем запрос для вывода сайтов, требующих проверки.
7. Если возникли ошибки, то выводим их описание.
8. Делаем запрос на выборку данных.
9. Проверяем корректность номера страницы и рассчитываем начальную и конечную позицию.

10. Выводим начало формы для удаления или одобрения сайта.
11. Выводим описания сайтов. Рядом с каждым описанием добавляем поле для установки флажка. Название сайта служит ссылкой на страницу редактирования описания сайта (catalog.pl).
12. Если число сайтов больше заданного максимального количества сайтов на странице, то выводим номера страниц.
13. Выводим конец формы для удаления или одобрения сайта.
14. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.43.

Листинг 5.43. Содержимое файла /cgi-bin/admin/moder.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Администрирование";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
```

```
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_admin();
   exit();
}
$db->do("SET NAMES cp1251");

print "<TABLE align=\"center\" width=\"600\" bgcolor=\"#FFFFFF\" ";
print "border=\"0\" cellpadding=\"2\" cellspacing=\"0\">\n";
print "<TR><TD align=\"left\" bgcolor=\"#FFFFFF\">\n\n";
print "<H1>Сайты на модерации</H1><BR>\n";

# Удаляем сайт
if (defined(param('del_msg')) && defined(param('check'))) {
   my @ch = param('check');
   my $count_ch = scalar(@ch);
   if ($count_ch > 0) {
      for (my $k=0; $k<$count_ch; $k++) {
         if ($ch[$k] =~ /^[0-9]+$/) {
            my $query_del = "DELETE FROM site ";
            $query_del .= "WHERE id_site=$ch[$k] LIMIT 1";
            $db->do($query_del);
         }
      }
   }
}

# Ставим флажок "Одобрено"
if (defined(param('upd_msg')) && defined(param('check'))) {
   my @ch = param('check');
   my $count_ch = scalar(@ch);
```

```
    if ($count_ch > 0) {
        for (my $k=0; $k<$count_ch; $k++) {
            if ($ch[$k] =~ /^[0-9]+$/) {
                my $query_upd = "UPDATE site SET status_site='y' ";
                $query_upd .= "WHERE id_site=$ch[$k] LIMIT 1";
                $db->do($query_upd);
            }
        }
    }
}

my $page = param('page') || 1;
my $query = "";
my $err_text = "";
my $text = "";

if (defined(param('search'))) {
    # Выводим поисковый запрос
    $text = param('search') || "";
    # Удаляем пробельные символы
    {
        $text =~ s/\r//g;
        local $/ = "";
        chomp($text);
    }
    $text =~ s/\n/ /g;
    $text =~ s/\t/ /g;
    $text =~ s/"/&quot;/g;
    if ($text eq "") {
        $err_text .= "Не задана строка поиска<BR>";
    }
    if ($err_text eq "") {
        # Добавляем защитные слэши перед спецсимволами
        $text =~ s/\\/\\\\/g;
        $text =~ s/'/\\'/g;
        $query = "SELECT site.id_site, site.url_site, site.title, ";
        $query .= "site.deskr, rubr.name_rubr FROM site, rubr ";
```

```
      $query .= "WHERE site.id_rubr=rubr.id_rubr AND ";
      $query .= "url_site LIKE '%$text%'";
   }
}

if ($query eq "") {
   # Выводим сайты для модерации
   $query = "SELECT site.id_site, site.url_site, site.title, ";
   $query .= "site.deskr, rubr.name_rubr FROM site, rubr ";
   $query .= "WHERE site.id_rubr=rubr.id_rubr AND ";
   $query .= "site.status_site='n' ORDER BY id_site";
}
if ($err_text ne "") {
   print "<DIV class=\"err\">$err_text</DIV><BR>\n";
}
# Выполняем запрос
my $count = 0;
my $all_array;
my $result = $db->prepare($query);
$result->execute();
if (!$result->err) {
   $all_array = $result->fetchall_arrayref();
   $count = scalar(@$all_array);
}

# Выводим сайты
my $count_page = 0;
if ($count =~ /^[0-9]+$/ && $count > 0) {
   $page = 1 if (!defined($page) || $page < 1);
   $page = 1 if ($page !~ /^[0-9]+$/);
   $count_page = $count/$DATA{'count_pos_page_site'};
   my $count_page2 = int($count_page);
   if ($count_page > $count_page2) {
      $count_page = $count_page2 + 1;
   }
   $page = $count_page if ($page > $count_page);
   my $pos_end = $page * $DATA{'count_pos_page_site'};
```

```
my $pos_start = $pos_end - $DATA{'count_pos_page_site'};
$pos_end = $count if ($pos_end > $count);
$pos_start = 0 if ($pos_start < 0);
print "<FORM>\n";
print "<DIV align=\"center\">";
print "<INPUT type=\"hidden\" name=\"page\" value=\"$page\">\n";
print "<INPUT type=\"submit\" name=\"upd_msg\" value=\"Одобрить\">\n";
print "<INPUT type=\"submit\" name=\"del_msg\" value=\"Удалить\">";
print "</DIV><BR>\n";
for (my $i = $pos_start; $i < $pos_end; $i++) {
   my $id_site = $all_array->[$i]->[0];
   my $url_site = $all_array->[$i]->[1];
   my $name_rubr = $all_array->[$i]->[4];
   my $title_site = $all_array->[$i]->[2];
   my $deskr_site = $all_array->[$i]->[3];
   print "<TABLE width=\"100%\" align=\"center\" border=\"0\" ";
   print "cellspacing=\"0\">\n";
   print "<TR><TD class=\"color_table\">\n";
   print "<INPUT type=\"checkbox\" name=\"check\" ";
   print "value=\"$id_site\"> \n";
   print "<A href=\"" . $DATA{'URL_SITE'};
   print "cgi-bin/admin/catalog.pl?id=$id_site\" ";
   print "target=\"_blank\">$title_site</A></TD></TR>\n";
   print "<TR><TD>\n";
   print "$deskr_site\n<BR>";
   print "<SPAN class=\"bold\">Рубрика: $name_rubr<BR>\n";
   print "URL сайта: </SPAN><A href=\"$url_site\" ";
   print "target=\"_blank\">$url_site</A>\n";
   print "</TD></TR></TABLE><BR>\n";
}
# Выводим количество страниц
if ($count > $DATA{'count_pos_page_site'}) {
   print "<SPAN class=\"bold\">Страницы:</SPAN>\n";
   if (defined(param('search'))) {
      for (my $j = 1; $j < $count_page + 1; $j++) {
         print "<A href=\"?page=$j&search=$text\">$j</A> \n";
      }
   }
```

```
      else {
         for (my $j = 1; $j < $count_page + 1; $j++) {
           print "<A href=\"?page=$j\">$j</A> \n";
         }
      }
   }
   print "</FORM>\n";
}
else {
   print "<DIV align=\"center\" class=\"bold\">";
   print "Сайтов на модерации нет</DIV><BR>\n";
}

print "</TD></TR></TABLE>\n";
# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_admin();
```

5.15.3. Редактирование описания произвольного сайта

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. Если форма для изменения описания сайта отправлена (параметр `param('go')` будет существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в переменных;
   - заменяем спецсимволы на HTML-эквиваленты;
   - добавляем защитные слэши перед спецсимволами MySQL;
   - проверяем корректность введенных данных;
   - если ошибок нет, то изменяем описание сайта.
3. Если возникли ошибки, то выводим их описание.
4. Если существует сообщение об удачном изменении описания сайта, то выводим его.

5. Если существует идентификатор сайта (параметр `param('id')` будет существовать):
   - с помощью метода `param()` получаем переданный идентификатор сайта и сохраняем его в переменной `$id`;
   - проверяем идентификатор сайта, и если это число, то делаем запрос;
   - выводим заполненную форму.
6. Если возникли ошибки, то выводим их описание.
7. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.44.

Листинг 5.44. Содержимое файла /cgi-bin/admin/catalog.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use URI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Администрирование";
# Описание страницы
my $description = "";
```

```
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_admin();
   exit();
}
$db->do("SET NAMES cp1251");

print "<H1>Администрирование каталога</H1><BR>\n";
print <<SCRIPT_COD;
<SCRIPT language="JavaScript">
<!--
function f_submit() {
   var m_frm=document.frm;
   if (m_frm.urls.value.length<8 || m_frm.titles.value=="" ||
      m_frm.deskr.value=="") {
      window.alert("Поле не заполнено");
      return false;
   }
   if (m_frm.titles.value.length>70) {
      var m_msg = "Длина Названия не должна превышать 70 символов";
      window.alert(m_msg);
      return false;
   }
   c_Reg = /^http:\\/\\/(www\\.)?([a-z0-9\\-]+\\.)+[a-z]{2,4}\$/i;
   if (!c_Reg.test(m_frm.urls.value)) {
      var m_m = "Недопустимый URL\\nПравильно — ";
      m_m += "\\"http://www.mail.ru\\"\\nНеправильно — ";
      m_m += "\\"http://www.mail.ru/\\""
```

```
      window.alert(m_m);
      m_frm.urls.focus();
      return false;
   }
   if (m_frm.rubr.options[m_frm.rubr.selectedIndex].value==0) {
      window.alert("Рубрика не выбрана");
      return false;
   }
return true;
}
//-->
</SCRIPT>
SCRIPT_COD

my $err_enter = "";
my $err_add = "";
my $ok_add = "";

if (defined(param('go'))) { # Если форма отправлена
   my $id_site = param('id') || 0;
   my $urls = param('urls') || "";
   my $rubr = param('rubr') || 0;
   my $titles = param('titles') || "";
   my $deskr = param('deskr') || "";
   my $hosts = param('hosts') || "";
   my $iq_site = param('iq_site') || 0;
   my $status = param('status_site') || 0;
   my $status_site = "";

   # Заменяем спецсимволы
   $titles =~ s/&/&amp;/g;
   $titles =~ s/</&lt;/g;
   $titles =~ s/>/&gt;/g;
   $titles =~ s/"/&quot;/g;
   $titles =~ s/\r//g;
   $titles =~ s/\t/ /g;
   $titles =~ s/\\/\\\\/g;
   $titles =~ s/'/\\'/g;
```

```
$deskr =~ s/&/&amp;/g;
$deskr =~ s/</&lt;/g;
$deskr =~ s/>/&gt;/g;
$deskr =~ s/"/&quot;/g;
$deskr =~ s/\r//g;
$deskr =~ s/\t/ /g;
$deskr =~ s/\\/\\\\/g;
$deskr =~ s/'/\\'/g;
# Удаляем символы новой строки
{
   local $/ = "";
   chomp($titles);
   chomp($deskr);
}
$titles =~ s/\n/ /g;
$deskr =~ s/\n/ /g;

if ($id_site !~ /^[0-9]+$/ || $id_site == 0) {
   $err_add .= "Идентификатор имеет неправильный формат<BR>";
}
else {
   my $query_site = "SELECT * FROM site WHERE id_site=$id_site";
   my $res_site = $db->prepare($query_site);
   $res_site->execute();
   unless (!$res_site->err && $res_site->rows() == 1) {
      $err_add .= "Сайт с таким id не найден<BR>";
   }
   $res_site->finish();
}
if ($iq_site !~ /^[0-9]+$/) {
   $err_add .= "Параметр iq_site имеет неправильный формат<BR>";
}
if ($rubr !~ /^[0-9]+$/ || $rubr == 0) {
   $err_add .= "Рубрика не выбрана или имеет неправильный формат<BR>";
}
else {
   my $query_rubr = "SELECT * FROM rubr WHERE id_rubr=$rubr";
   my $res_rubr = $db->prepare($query_rubr);
```

```
    $res_rubr->execute();
    unless (!$res_rubr->err && $res_rubr->rows() == 1) {
       $err_add .= "Рубрика не найдена<BR>";
    }
    $res_rubr->finish();
}
if (length($titles) > 70 || length($deskr) > 500) {
    $err_add .= "Длина Названия ресурса не должна превышать ";
    $err_add .= "70 символов, а Описания — 500<BR>";
}
 if ($titles =~ /[a-za-я0-9,\.\"\']{26}/i) {
    $err_add .= "Поле Название ресурса содержит более 25 ";
    $err_add .= "символов без пробела!<BR>\n";
 }
 if ($deskr =~ /[a-za-я0-9,\.\"\']{26}/i) {
    $err_add .= "Поле Описание содержит более 25 символов ";
    $err_add .= "без пробела!<BR>\n";
 }
if (length($titles) < 2 || length($deskr) < 2) {
    $err_add .= "Не заполнено обязательное поле<BR>";
}
my $host;
if ($urls !~ /^http:\/\/(www\.)?([a-z0-9\-]+\.)+[a-z]{2,4}$/i ||
     length($urls)>200)
{
    $err_add .= "Недопустимый URL !<BR>";
}
else {
    $urls = lc($urls);
    my $url = URI->new($urls);
    $host = $url->host();
    if ($host =~ /^www\./) {
       $host = substr($host, 4);
    }
}
if ($hosts ne $host) {
    $err_add .= "Не совпадают URL и хост<BR>";
}
```

```
   if ($status == 1) { $status_site = 'y'; }
   elsif ($status == 2) { $status_site = 'n'; }
   elsif ($status == 3) { $status_site = 's'; }
   else {
      $err_add .= "Статус сайта имеет неправильный формат<BR>";
   }

   if ($err_add eq "") {
      # Если ошибок нет, пробуем обновить информацию
      my $query = "UPDATE site SET id_rubr=$rubr, ";
      $query .= "url_site='$urls', url_site_dop='$hosts', ";
      $query .= "title='$titles', deskr='$deskr', ";
      $query .= "status_site='$status_site', iq_site=$iq_site ";
      $query .= "WHERE id_site=$id_site LIMIT 1";
      my $count = $db->do($query);
      if ($count == 1) {
         $ok_add="Информация о сайте $urls обновлена<BR>";
      }
      else {
         $err_add .= "Попробуйте сделать запрос через ";
         $err_add .= "некоторое время<BR>";
      }
   }
}

if ($err_add ne "") {
   print "<DIV class=\"err\">$err_add</DIV><BR>\n";
}
if ($ok_add ne "") {
   print "<DIV class=\"ok\">$ok_add</DIV><BR>\n";
}

if (defined(param('id'))) {
   # Выводим заполненную форму
   my $id = param('id') || 0;
   if ($id !~ /^[0-9]+$/ || $id == 0) {
      $err_enter .= " Недопустимые символы в параметре id<BR>";
   }
```

```
else {
   my $query_enter = "SELECT * FROM site WHERE id_site=$id";
   my $res_enter = $db->prepare($query_enter);
   $res_enter->execute();
   unless (!$res_enter->err && $res_enter->rows() == 1) {
      $err_enter .= "Сайт не найден<BR>";
   }
   else {
      my @row = $res_enter->fetchrow_array();
      my $id_site = $row[0];
      my $rubr = $row[1];
      my $urls = $row[3];
      my $hosts = $row[4];
      my $titles = $row[5];
      my $deskr = $row[6];
      my $status_site = $row[7];
      my $iq_site = $row[8];

      print "<TABLE width=\"600\" align=\"center\" border=\"0\" ";
      print "cellspacing=\"0\">\n";
      print "<TR><TD class=\"color_table\">\n";
      print "<A href=\"$urls\" target=\"_blank\">";
      print "$titles</A></TD></TR>\n";
      print "<TR><TD>\n";
      print "$deskr\n</TD></TR></TABLE><BR><BR>\n";
      print "<FORM action=\"catalog.pl\" method=\"POST\" ";
      print "name=\"frm\" id=\"frm\" ";
      print "onsubmit=\"return f_submit();\">\n";
      print "<TABLE width=\"100%\" border=\"0\" cellpadding=\"1\" ";
      print "align=\"center\">\n";
      print "<TR><TD align=\"right\" width=\"40%\">\n";
      print "<SPAN class=\"bold\">URL: </SPAN>\n";
      print "</TD><TD>\n";
      print "<INPUT type=\"text\" name=\"urls\" id=\"urls\" ";
      print "size=\"40\" value=\"$urls\" class=\"txt_frm\">\n";
      print "<INPUT type=\"hidden\" name=\"id\" ";
      print "value=\"$id_site\">\n";
```

```
print "</TD></TR>\n";
print "<TR><TD align=\"right\" width=\"40%\">\n";
print "<SPAN class=\"bold\">Хост: </SPAN>\n";
print "</TD><TD>\n";
print "<INPUT type=\"text\" name=\"hosts\" id=\"hosts\" ";
print "size=\"40\" value=\"$hosts\" class=\"txt_frm\">\n";
print "</TD></TR>\n";
print "</TABLE><BR>\n";
print "<DIV align=\"center\">\n";
print "<SELECT name=\"rubr\" id=\"rubr\" ";
print "class=\"select_frm\">\n";
print "<OPTION value=\"0\">----------------------";
print "Выберите рубрику-----------------</OPTION>\n";

my $query_rubr = "SELECT * FROM rubr ORDER BY name_rubr";
my $res_rubr = $db->prepare($query_rubr);
$res_rubr->execute();
if (!$res_rubr->err) {
   while (my @row_rubr = $res_rubr->fetchrow_array()) {
      print "<OPTION value=\"$row_rubr[0]\"";
      if ($rubr == $row_rubr[0]) {
         print " selected>";
      }
      else {
         print ">";
      }
      $row_rubr[1] =~ s/"/&quot;/g;
      print $row_rubr[1];
      print "</OPTION>\n";
   }
}
$res_rubr->finish();

print "</SELECT>\n";
print "</DIV><BR>\n";
print "<TABLE width=\"100%\" border=\"0\" cellpadding=\"1\" ";
print "align=\"center\">\n";
print "<TR><TD align=\"right\" width=\"40%\">\n";
```

```
print "<SPAN class=\"bold\">Название ресурса: </SPAN>\n";
print "</TD><TD>\n";
print "<INPUT type=\"text\" name=\"titles\" id=\"titles\" ";
print "size=\"40\" maxlength=\"70\" class=\"txt_frm\" ";
print "value=\"$titles\">\n";
print "</TD></TR>\n";
print "<TR><TD align=\"right\" width=\"40%\">";
print "<SPAN class=\"bold\">Описание ресурса: </SPAN>\n";
print "</TD><TD>\n";
print "<TEXTAREA name=\"deskr\" id=\"deskr\" ";
print "class=\"textarea_frm\" ";
print "style=\"width: 300px; height: 150px\">";
print $deskr . "</TEXTAREA>\n";
print "</TD></TR>\n";
print "<TR><TD align=\"right\" width=\"40%\">\n";
print "<SPAN class=\"bold\">IQ-сайта: </SPAN>\n";
print "</TD><TD>\n";
print "<INPUT type=\"text\" name=\"iq_site\" id=\"iq_site\" ";
print "size=\"40\" class=\"txt_frm\" value=\"$iq_site\">\n";
print "</TD></TR>\n";
print "<TR><TD align=\"right\" width=\"40%\">\n";
print "<SPAN class=\"bold\">Статус сайта: </SPAN>\n";
print "</TD><TD>\n";
print "<INPUT type=\"radio\" name=\"status_site\" ";
print "id=\"status_site\" value=\"1\"";
print " checked" if ($status_site eq 'y');
print "><SPAN class=\"bold\">Активен</SPAN><BR>\n";
print "<INPUT type=\"radio\" name=\"status_site\" ";
print "id=\"status_site\" value=\"2\"";
print " checked" if ($status_site eq 'n');
print "><SPAN class=\"bold\">На модерации</SPAN><BR>\n";
print "<INPUT type=\"radio\" name=\"status_site\" ";
print "id=\"status_site\" value=\"3\"";
print " checked" if ($status_site eq 's');
print "><SPAN class=\"bold\">Временно не работает</SPAN>\n";
print "</TD></TR>\n";
print "<TR><TD align=\"right\" width=\"40%\">.:</TD><TD>\n";
```

```
            print "<INPUT type=\"submit\" name=\"go\" ";
            print "value=\"Изменить описание сайта\" class=\"txt_frm\">\n";
            print "</TD></TR>\n";
            print "</TABLE>\n";
            print "</FORM><BR>\n";
        }
    }
}

if ($err_enter ne "") {
    print "<DIV class=\"err\">$err_enter</DIV><BR>\n";
}

# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_admin();
```

5.15.4. Администрирование гостевой книги

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. С помощью метода `param()` получаем переданный номер страницы и сохраняем его в переменной `$page`.
3. Если форма для удаления сообщения отправлена (параметры `param('del_msg')` и `param('check')` будут существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в массиве `@ch`;
   - в цикле проверяем идентификатор сообщения и, если это число, удаляем сообщение.
4. Если форма для одобрения сообщения отправлена (параметры `param('upd_msg')` и `param('check')` будут существовать):
   - с помощью метода `param()` получаем переданные параметры и сохраняем их в массиве `@ch`;
   - в цикле проверяем идентификатор сообщения и, если это число, изменяем статус сообщения на "прочитано".

5. Определяем количество сообщений.
6. Проверяем корректность номера страницы и рассчитываем начальную и конечную позицию.
7. Выводим начало формы для удаления или одобрения сообщения.
8. Выводим сообщения. Рядом с каждым сообщением добавляем поле для установки флажка.
9. Если сообщение имеет статус "не прочитано", то перед сообщением выводим "NEW !!!".
10. Если число сообщений больше заданного максимального количества сообщений на странице, то выводим номера страниц.
11. Выводим конец формы для удаления или одобрения сообщения.
12. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.45.

Листинг 5.45. Содержимое файла /cgi-bin/admin/gbook.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin &f_date_gm );

print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Администрирование";
```

```
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords, 2);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_admin();
   exit();
}
$db->do("SET NAMES cp1251");

print "<TABLE align=\"center\" width=\"600\" bgcolor=\"#FFFFFF\" ";
print "border=\"0\" cellpadding=\"2\" cellspacing=\"0\">\n";
print "<TR><TD align=\"left\" bgcolor=\"#FFFFFF\">\n\n";
print "<H1>Гостевая книга</H1><BR>\n";
my $page = param('page') || 1;

# Удаляем сообщение
if (defined(param('del_msg')) && defined(param('check'))) {
   my @ch = param('check');
   my $count_ch = scalar(@ch);
   if ($count_ch > 0) {
      for (my $k=0; $k<$count_ch; $k++) {
         if ($ch[$k] =~ /^[0-9]+$/) {
            my $query_del = "DELETE FROM gbook ";
            $query_del .= "WHERE id_msg=$ch[$k] LIMIT 1";
            $db->do($query_del);
         }
      }
   }
}
```

```
# Ставим флажок "Прочитано"
if (defined(param('upd_msg')) && defined(param('check'))) {
   my @ch = param('check');
   my $count_ch = scalar(@ch);
   if ($count_ch > 0) {
      for (my $k=0; $k<$count_ch; $k++) {
         if ($ch[$k] =~ /^[0-9]+$/) {
            my $query_upd = "UPDATE gbook SET msg_new='y' ";
            $query_upd .= "WHERE id_msg=$ch[$k] LIMIT 1";
            $db->do($query_upd);
         }
      }
   }
}

# Определяем количество сообщений
my $count = 0;
my $query_count = "SELECT COUNT(*) FROM gbook";
my $res_count = $db->prepare($query_count);
$res_count->execute();
if (!$res_count->err && $res_count->rows() == 1) {
   my @row_count = $res_count->fetchrow_array();
   $count = $row_count[0];
}
$res_count->finish();

# Выводим сообщения
if ($count =~ /^[0-9]+$/ && $count > 0) {
   $page = 1 if (!defined($page) || $page < 1);
   $page = 1 if ($page !~ /^[0-9]+$/);
   my $count_page = $count/$DATA{'count_pos_page_gbook'};
   my $count_page2 = int($count_page);
   if ($count_page > $count_page2) {
      $count_page = $count_page2 + 1;
   }
   $page = $count_page if ($page > $count_page);
   print "<FORM>\n";
```

```
print "<DIV align=\"center\"><INPUT type=\"hidden\" ";
print "name=\"page\" value=\"$page\">\n";
print "<INPUT type=\"submit\" name=\"upd_msg\" ";
print "value=\"Прочитано\">\n";
print "<INPUT type=\"submit\" name=\"del_msg\" ";
print "value=\"Удалить\"></DIV><BR>\n";
my $pos_start = ($page — 1) * $DATA{'count_pos_page_gbook'};
my $q = "SELECT author, msg, ";
$q .= "DATE_FORMAT(msg_date, '%d.%m.%Y %H:%i'), ";
$q .= "id_msg, msg_new ";
$q .= "FROM gbook ORDER BY id_msg DESC ";
$q .= "LIMIT $pos_start, $DATA{'count_pos_page_gbook'}";
my $res = $db->prepare($q);
$res->execute();
if (!$res->err && $res->rows() > 0) {
   while (my @row = $res->fetchrow_array()) {
      # Выводим описание
      my $id = $row[3];
      my $author = $row[0];
      my $msg = $row[1];
      my $status_msg = $row[4];
      my $msg_date = $row[2];
      print "<TABLE width=\"100%\" align=\"center\" ";
      print "border=\"0\" cellspacing=\"0\">\n";
      print "<TR><TD class=\"color_table\">\n";
      print "<INPUT type=\"checkbox\" name=\"check\" value=\"$id\"> ";
      print "<B>$msg_date $author</B></TD></TR>\n";
      print "<TR><TD>\n";
      if ($status_msg eq "n") {
         print "<SPAN class=\"err\">NEW !!! </SPAN> \n";
      }
      print "$msg\n</TD></TR></TABLE><BR>\n";
   }
}
else { print "Ошибка"; }
$res->finish();
```

```
   # Выводим количество страниц
   if ($count > $DATA{'count_pos_page_gbook'}) {
      &f_table_page_start(); # Таблица для количества страниц Начало
      print "<SPAN class=\"bold\">Страницы:</SPAN>\n";
      for (my $j = 1; $j < $count_page + 1; $j++) {
         if ($j == $page) {
            print "[$j] \n";
         }
         else {
            print "<A href=\"?page=$j\">$j</A> \n";
         }
      }
      &f_table_page_end(); # Таблица для количества страниц Конец
   }
    print "</FORM>\n";
}
else {
   print "<DIV align=\"center\" class=\"bold\">";
   print "Сообщений нет</DIV><BR>\n";
}
print "</TD></TR></TABLE>\n";

# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_admin();
```

Каждое новое сообщение содержит "NEW !!!". Для того чтобы сбросить флаг, необходимо установить флажки нужных сообщений и нажать кнопку **Прочитано**. Надписи исчезнут. Для удаления сообщения нужно установить флажок напротив сообщения и нажать **Удалить**.

5.15.5. Создание страницы статистики

Данная страница будет первой, которую увидит администратор при входе в Личный кабинет. По этой причине она должна содержать информацию о количестве сайтов и сообщений, требующих проверки, а также все сообщения об ошибках.

Алгоритм работы программы следующий:

1. Подключаемся к базе данных. Если подключиться не удалось, выводим сообщение и завершаем работу скрипта.
2. Если форма для очистки файла errlog.txt отправлена (параметр `param('goErr')` будет существовать), то записываем в файл пробел.
3. Если форма для очистки файла err401.txt отправлена (параметр `param('go401')` будет существовать), то записываем в файл пробел.
4. Если форма для очистки файла err403.txt отправлена (параметр `param('go403')` будет существовать), то записываем в файл пробел.
5. Если форма для очистки файла err404.txt отправлена (параметр `param('go404')` будет существовать), то записываем в файл пробел.
6. Делаем запрос на общее количество зарегистрированных сайтов.
7. Делаем запрос на количество сайтов с группировкой по статусу.
8. Делаем запрос на общее количество сообщений в гостевой книге.
9. Делаем запрос на количество непрочитанных сообщений.
10. Выводим статистику.
11. Считываем содержимое файлов errlog.txt, err404.txt, err401.txt и err403.txt в соответствующие поля для многострочного текста и выводим кнопки для очистки этих файлов.
12. Закрываем соединение с базой данных.

Исходный код программы приведен в листинге 5.46.

Листинг 5.46. Содержимое файла /cgi-bin/admin/index.pl

```
#!/usr/bin/perl
BEGIN {
   use CGI::Carp qw( carpout );
   open(ERRLOG, ">>../errlog.txt") or die("Ошибка\n");
   carpout(\*ERRLOG);
}
use strict;
use DBI;
use CGI qw( :standard );
# Задаем местонахождение файла конфигурации
use lib "/home/site.ru/cgi-bin/config";
# Подключаем файл конфигурации
use Allscript qw( :DEFAULT :admin &f_date_gm );
```

```
print "Expires: Sun, 27 May 2007 01:00:00 GMT\n";
print "Last-Modified: " . &f_date_gm() ." GMT\n";
print "Cache-Control: no-store, no-cache, must-revalidate\n";
print "Pragma: no-cache\n";
print "Content-type: text/html; charset=windows-1251\n\n";

# Заголовок
my $title = "Администрирование";
# Описание страницы
my $description = "";
# Ключевые слова для поисковых машин
my $keywords = "";
# Выводим верхний колонтитул
&f_header_all($title, $description, $keywords);

# Подключаемся к базе данных
my $ds = 'DBI:mysql:' . $DATA{'DB'} . ':' . $DATA{'HOST'};
my $db = DBI->connect($ds, $DATA{'LOGIN'}, $DATA{'PASSW'});
if (!$db) {
   print "<DIV class=\"err\">Не удалось установить ";
   print "соединение с базой данных</DIV>";
   &f_footer_admin();
   exit();
}
$db->do("SET NAMES cp1251");

if (defined(param('goErr'))) {
   open(FILEERR, '>../errlog.txt') or die("Ошибка $!");
   print FILEERR " ";
   close(FILEERR);
}
if (defined(param('go401'))) {
   open(FILE401, '>../err401.txt') or die("Ошибка $!");
   print FILE401 " ";
   close(FILE401);
}
if (defined(param('go403'))) {
   open(FILE403, '>../err403.txt') or die("Ошибка $!");
```

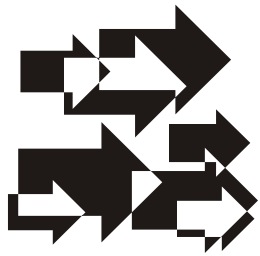
```
   print FILE403 " ";
   close(FILE403);
}
if (defined(param('go404'))) {
   open(FILE404, '>../err404.txt') or die("Ошибка $!");
   print FILE404 " ";
   close(FILE404);
}

# Определяем количество сайтов в каталоге
my $count_site = 0;
my $query1 = "SELECT COUNT(*) FROM site";
my $result1 = $db->prepare($query1);
$result1->execute();
if (!$result1->err) {
   if ($result1->rows() == 1) {
      my @row1 = $result1->fetchrow_array();
      $count_site = $row1[0];
   }
}
$result1->finish();

# Определяем количество сайтов по категориям
my %Mass = ("y" => 0, "n" => 0, "s" => 0);
my $query2 = "SELECT status_site, COUNT(id_site) ";
$query2 .= "FROM site GROUP BY status_site";
my $result2 = $db->prepare($query2);
$result2->execute();
if (!$result2->err) {
   while (my @row2 = $result2->fetchrow_array()) {
      $Mass{$row2[0]} = $row2[1];
   }
}
$result2->finish();

# Определяем количество сообщений в гостевой книге
my $count_msg = 0;
my $query3 = "SELECT COUNT(*) FROM gbook";
```

```
my $result3 = $db->prepare($query3);
$result3->execute();
if (!$result3->err) {
   if ($result3->rows() == 1) {
      my @row3 = $result3->fetchrow_array();
      $count_msg = $row3[0];
   }
}
$result3->finish();

# Определяем количество непрочитанных сообщений
my $msg_new = 0;
my $query4 = "SELECT COUNT(id_msg) FROM gbook WHERE msg_new='n'";
my $result4 = $db->prepare($query4);
$result4->execute();
if (!$result4->err) {
   if ($result4->rows() == 1) {
      my @row4 = $result4->fetchrow_array();
      $msg_new = $row4[0];
   }
}
$result4->finish();

print "<H1>Статистика</H1><BR>\n";
print "<BR><DIV align=\"center\" class=\"bold\">";
print "Количество сайтов в каталоге — $count_site<BR>";
print "- активных — $Mass{'y'}<BR>";
print "<A href=\"moder.pl\">- на модерации — $Mass{'n'}</A><BR>";
print "- нерабочие — $Mass{'s'}<BR>";
print "</DIV>";
print "<BR><DIV align=\"center\" class=\"bold\">";
print "Количество сообщений в гостевой книге — $count_msg<BR>";
print "<A href=\"gbook.pl\">- не прочитано — $msg_new</A>";
print "</DIV>";

print "<BR><DIV align=\"center\" class=\"bold\">ErrorLog<BR>\n";
print "<TEXTAREA name=\"msgErr\" id=\"msgErr\" class=\"textarea_frm\" ";
```

```
print "style=\"width: 500px; height: 150px\">\n";
open(ERR, '../errlog.txt') or die("Ошибка $!");
{
local $/ = "";
my $errtext = <ERR> || "";
$errtext =~ s/&/&amp;/g;
$errtext =~ s/</&lt;/g;
$errtext =~ s/>/&gt;/g;
$errtext =~ s/"/&quot;/g;
print $errtext;
}
close(ERR);
print "</TEXTAREA><BR>\n";
print "<FORM>\n";
print "<INPUT type=\"submit\" name=\"goErr\" value=\"Очистить\">\n";
print "</FORM></DIV>\n";
print "<BR><DIV align=\"center\" class=\"bold\">Ошибка 404<BR>\n";
print "<TEXTAREA name=\"msg404\" id=\"msg404\" class=\"textarea_frm\" ";
print "style=\"width: 500px; height: 150px\">\n";
open(ERR404, '../err404.txt') or die("Ошибка $!");
{
local $/ = "";
print <ERR404>;
}
close(ERR404);
print "</TEXTAREA><BR>\n";
print "<FORM>\n";
print "<INPUT type=\"submit\" name=\"go404\" value=\"Очистить\">\n";
print "</FORM></DIV>\n";
print "<BR><DIV align=\"center\" class=\"bold\">Ошибка 401<BR>\n";
print "<TEXTAREA name=\"msg401\" id=\"msg401\" class=\"textarea_frm\" ";
print "style=\"width: 500px; height: 150px\">\n";
open(ERR401, '../err401.txt') or die("Ошибка $!");
{
local $/ = "";
print <ERR401>;
}
```

```
close(ERR401);
print "</TEXTAREA><BR>\n";
print "<FORM>\n";
print "<INPUT type=\"submit\" name=\"go401\" value=\"Очистить\">\n";
print "</FORM></DIV>\n";
print "<BR><DIV align=\"center\" class=\"bold\">Ошибка 403<BR>\n";
print "<TEXTAREA name=\"msg403\" id=\"msg403\" class=\"textarea_frm\" ";
print "style=\"width: 500px; height: 150px\">\n";
open(ERR403, '../err403.txt') or die("Ошибка $!");
{
local $/ = "";
print <ERR403>;
}
close(ERR403);
print "</TEXTAREA><BR>\n";
print "<FORM>\n";
print "<INPUT type=\"submit\" name=\"go403\" value=\"Очистить\">\n";
print "</FORM></DIV>\n";

# Закрываем соединение с базой данных
$db->disconnect();
# Выводим нижний колонтитул
&f_footer_admin();
```

ГЛАВА 6



Публикация сайта

6.1. Выбор тарифного плана

В качестве виртуального хостинга рассмотрим хостинг Majordomo (**http://majordomo.ru/**). Хостинг предлагает множество тарифных планов, от простой парковки домена до размещения своего сервера. Итак, определимся с нашими требованиями к хостингу и выберем тарифный план.

Для нормальной работы сайта необходима поддержка следующих технологий:

- операционная система семейства UNIX;
- Web-сервер Apache;
- поддержка PHP и Perl;
- возможность использования баз данных MySQL;
- достаточно места под сайт.

С тарифными планами хостинга можно ознакомиться на странице **http://majordomo.ru/hosting/**. Практически все тарифные планы отвечают первым трем требованиям. Поддержка MySQL есть только в трех (Соло+, Мастер и Профи). Эти же тарифные планы отвечают всем остальным нашим требованиям.

Рассмотрим возможности трех выбранных тарифов:

- поддерживается PHP 4 и 5 версий, а также язык Perl;
- количество баз данных MySQL от 1 до 10;
- дисковое пространство от 250 Мбайт до 5 Гбайт;
- бонус для регистрации платного домена в зоне .RU на свое имя;

- неограниченный трафик;
- есть поддержка от 1 до 10 доменов;
- для администрирования баз данных есть программа phpMyAdmin;
- есть поддержка запуска программ в определенное время (сервис cron);
- невысокая стоимость;
- постоянный доступ к сайту;
- круглосуточная техническая поддержка.

На начальном этапе достаточно выбрать минимальный тариф (Соло+). По мере роста популярности сайта можно сменить тариф на Мастер или Профи, через панель управления, или арендовать сервер.

6.2. Регистрация аккаунта

Для начала регистрации переходим на страницу **http://majordomo.ru/order/facechoice.php**. На первом шаге выбираем, на кого будет оформлен хостинг — ставим флажок **Физическое лицо** и нажимаем кнопку **Продолжить оформление заказа**.

Отобразится страница **Публичная оферта о предоставлении услуг**. Для продолжения регистрации необходимо принять условия оферты. Устанавливаем флажок **Согласен** и нажимаем кнопку **Продолжить регистрацию**.

На следующем шаге выбираем тарифный план (например, **Мастер**) и вводим информацию о себе (рис. 6.1).

После заполнения всех полей, обозначенных звездочкой, нажимаем кнопку **Продолжить**. На указанный адрес будет выслано письмо с инструкцией по активации аккаунта.

> ***Обратите внимание***
>
> Если аккаунт не будет активирован в течение недели, то он будет удален.

При успешной активации будут предоставлены данные для входа в панель управления. Например:

```
Логин — ac_24734
Пароль — UAi6lyu6
```

> ***Обратите внимание***
>
> Логин и пароль необходимо сохранить.

Через некоторое время придет письмо с регистрационными данными, а также данные для доступа по протоколу FTP. Например:

```
IP-адрес:       78.108.81.170
login:          f24734
password:       UAi6lyu6
```

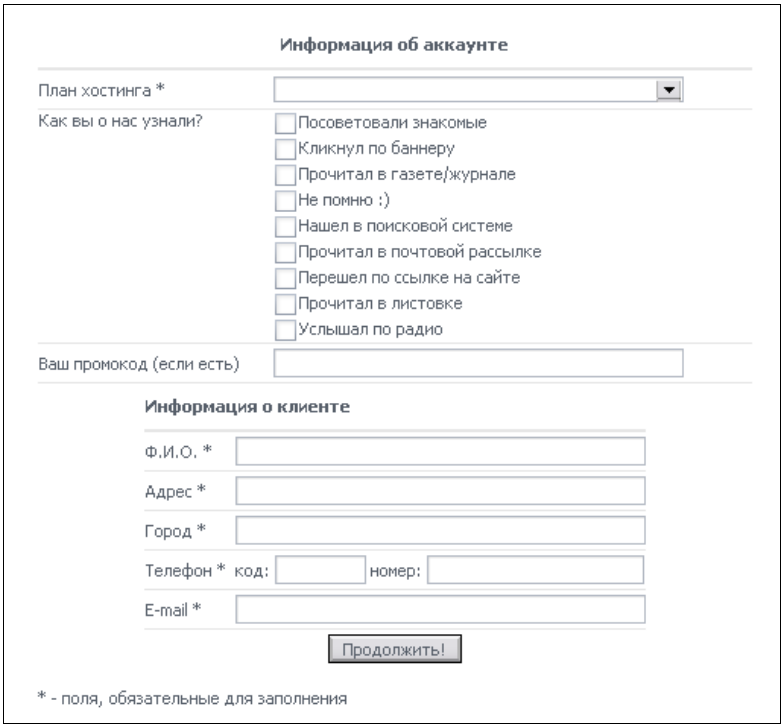


Рис. 6.1. Выбор тарифного плана

Сохраните эти данные. Они понадобятся нам при подключении к серверу по протоколу FTP. Настройку FTP-клиентов мы рассмотрим чуть позже.

Для входа в панель управления переходим по адресу **https://control.majordomo.ru/**. Вводим логин, пароль и нажимаем кнопку **Войти**. Перед нами панель управления. В верхней левой части окна находятся две ссылки: Аккаунт и Домены (рис. 6.2).

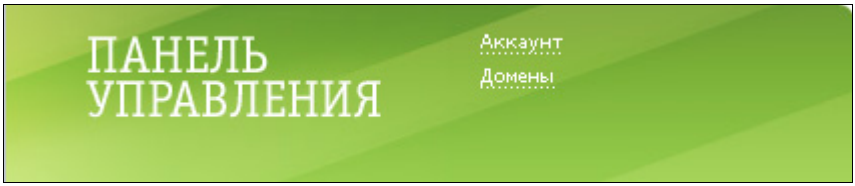


Рис. 6.2. Структура меню панели управления до оплаты

6.3. Регистрация доменного имени

Прежде чем заполнять заявку на регистрацию доменного имени, необходимо пополнить баланс. Переходим на страницу **Аккаунт | Пополнение баланса**. Выбираем наиболее подходящий способ и оплачиваем. После зачисления денег будут доступны дополнительные возможности панели управления (рис. 6.3).

Рис. 6.3. Структура меню панели управления после оплаты

Теперь можно приступить к регистрации домена. Переходим на страницу **Домены | Данные для регистрации**. Устанавливаем флажок **Для физического лица**. Откроется страница с формой для ввода данных (рис. 6.4).

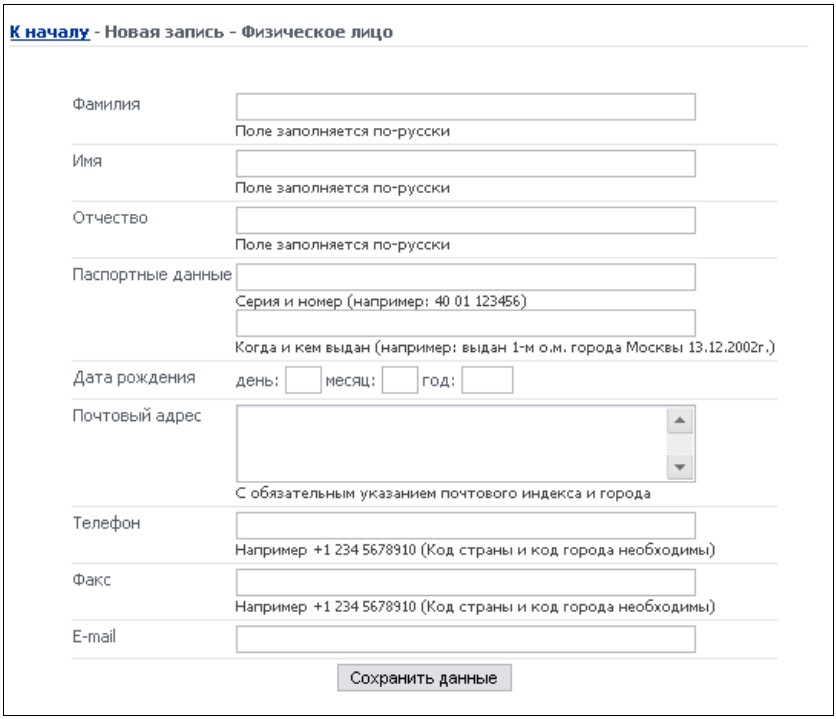


Рис. 6.4. Создание персоны для регистрации доменных имен

Заполняем форму и нажимаем кнопку **Сохранить данные**. Созданная персона отобразится в разделе **Физические лица** на странице **Домены | Данные для регистрации**. Обратите особое внимание на отсутствие ошибок в фамилии, имени и отчестве, а также правильность данных паспорта. Эти данные будут использованы для регистрации домена. В противном случае будете доказывать, что именно вы владелец домена.

Теперь можно подать заявку на регистрацию домена. Переходим на страницу **Домены | Заказ домена**. Отобразится форма, изображенная на рис. 6.5. Вводим название домена и выбираем зону. Устанавливаем флажки **Регистрация** и **Отдельный сайт**. Из списка **Владелец домена** выбираем созданную ранее запись. Нажимаем кнопку **Заказать**.

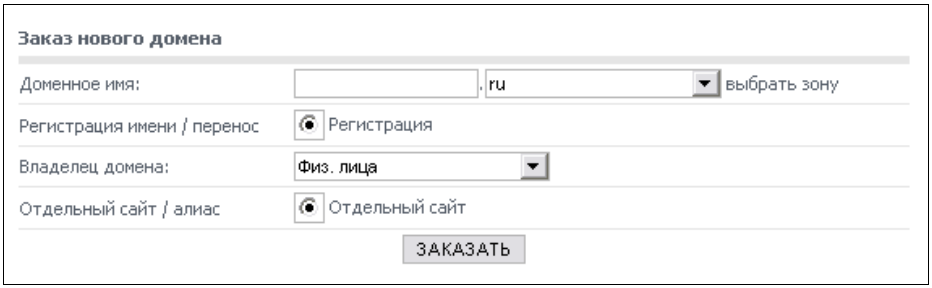


Рис. 6.5. Заказ нового домена

ОБРАТИТЕ ВНИМАНИЕ

Домен должен быть свободен. Для проверки необходимо проверить домен на странице **http://majordomo.ru/domain/**. Регистрировать домен можно только после получения статуса "свободен".

После подачи заявки придет письмо с указанием технического домена, его можно использовать для настройки сайта до момента регистрации вашего основного домена.

6.4. Структура панели управления

Рассмотрим структуру панели управления и основное назначение ее разделов:

- **Аккаунт** — в данном разделе расположена основная информация:
  - **Сводка** — содержит информацию о тарифном плане, состоянии счетов и количестве оплаченных дней, статистику использования места. Здесь же можно выбрать опцию представления хостинга в кредит на 14 дней при окончании средств на счете, а также указать, за сколько дней до

окончания хостинга необходимо присылать уведомление об окончании средств на контактный E-mail;

- **Смена пароля** — здесь можно изменить пароль для входа в панель управления, а также пароль доступа по FTP;
- **Информация о владельце** — на этой странице можно указать паспортные данные владельца аккаунта и адрес регистрации, контактные телефоны и дополнительные E-mail-адреса для связи;
- **Баланс и тарифный план** — здесь можно посмотреть полную стоимость аккаунта в месяц, а также изменить тарифный план;
- **Пополнение баланса** — позволяет пополнить баланс. Мы уже рассматривали этот пункт в *разд. 6.3*;
- **Заявленный платеж** — заполняя форму на этой странице, вы обязуетесь пополнить свой счет в течение 10 дней на указанную сумму. При наличии в системе заявленного платежа аккаунт продолжает работать даже при отрицательном балансе. Если указанная сумма не поступит на счет в течение 10 дней, то аккаунт будет заблокирован;
- **Логи авторизаций** — содержит записи всех заходов в панель управления и доступа по FTP;
- **Использование CPU** — на этой странице расположены три графика, которые показывают суммарную нагрузку на процессор сервера от всех ваших скриптов за определенный период. Первый график содержит данные за последние часы, а второй содержит данные за текущий день. Данные на третьем графике содержат значения за последние две недели. Обратите внимание, один аккаунт не может потреблять более 15 % ресурсов сервера. В случае превышения этого порога возможна временная блокировка аккаунта;
- **Использование MySQL** — на графике, расположенном на этой странице, показано использование MySQL-сервера скриптами в течение текущего дня, а на таблице под графиком показано количество запросов от скриптов к MySQL-серверу в течение текущего дня;
- **Ограничение доступа по FTP** — здесь можно ограничить доступ к серверу по протоколу FTP, разрешив вход только с одного или нескольких IP-адресов. Если не указан ни один адрес, доступ открыт с любого IP-адреса. Если указан хотя бы один или несколько адресов, доступ по FTP разрешен только с этих адресов;
- **Ограничение доступа к панели** — на этой странице можно ограничить доступ к панели управления, разрешив вход только с одного или нескольких IP-адресов. Если не указан ни один адрес, доступ к панели

открыт с любого IP-адреса. Если указан хотя бы один или несколько адресов, вход в панель разрешен только с этих адресов;

- **Почта** — здесь можно создать почтовый ящик, оформить пересылку почты на другой E-mail или воспользоваться Web-интерфейсом для чтения почты:
  - **Объем почты и квота** — здесь указывается общий объем почты на аккаунте и объем всех ящиков домена;
  - **Почтовые ящики** — на этой странице можно создать новый почтовый ящик, сменить пароль на существующем ящике, а также удалить почтовый ящик. Чтобы получать письма, приходящие на несуществующие адреса, необходимо создать почтовый ящик с названием postmaster. Почта в ящике postmaster хранится не более 2 месяцев, потом более старые письма автоматически удаляются. При поступлении более 100 писем в час прием почты для несуществующих ящиков автоматически прекращается. В качестве примера создадим этот почтовый ящик. В разделе **Создание почтового ящика** в поле **Адрес** вводим `postmaster`. Заполняем поле **Пароль** и нажимаем кнопку **Создать**. Чтобы создать пересылку, нажимаем на кнопку **Пересылать** на странице **Почта | Почтовые ящики**. Теперь все письма, присланные на несуществующие адреса, будут приходить на ящик postmaster;
  - **Переадресации** — в данном разделе можно создать пересылку всей корреспонденции на другой адрес, а также удалить существующую пересылку. Для переадресации из списка **Из:** выбираем название ящика, а в поле **В:** указываем список адресов получателей. После каждого адреса, кроме последнего, должна стоять запятая. Каждый адрес должен располагаться в новой строке. Если в качестве почтового ящика указана звездочка (например, *@perlbook.spb.ru), то на адрес пересылки будут отправлены все письма для данного домена, исключая те адреса, для которых явным образом прописаны переадресации. Обратите внимание, при использовании звездочки нельзя использовать опцию "оставлять копию";
  - **Защита от спама (RBL)** — здесь можно включить или выключить защиту от спама на доменах;
  - **Защита от спама (фильтры)** — данная услуга является частью услуги "Расширенное управление почтой", включающей в себя антивирусную проверку почты, дополнительные фильтры для почты и статистику **Awstats** для сайта;
  - **WebMail** — здесь можно работать с почтовым ящиком через Web-интерфейс;

- **Антивирус** — данная услуга является частью услуги "Расширенное управление почтой", включающей в себя антивирусную проверку почты, дополнительные фильтры для почты и статистику **Awstats** для сайта;
- **Расширенное управление** — здесь можно включить услугу "Расширенное управление почтой. Услуга является платной;
- **Статистика** — на этой странице указывается статистика по количеству писем;

❐ **Сайт** — в данном разделе расположены основные настройки сайта:

- **Индексные файлы** — здесь можно задать название файла, который будет выдаваться по умолчанию при запросах без указания конкретного файла (например, **http://perlbook.spb.ru/** или **http://perlbook.spb.ru/folder/**). Если индексный файл не найден, то выдается ошибка 403. По умолчанию индексными файлами являются index.html, index.htm и index.php. Если вы пользуетесь этими стандартными именами, то больше ничего указывать не нужно;
- **.htaccess и .htpasswd** — данный раздел позволяет создать файлы .htaccess и .htpasswd для защиты определенной папки средствами сервера Apache. Более подробно защиту папки и другие настройки мы рассмотрим чуть позже;
- **Кодировка сайта** — здесь можно указать, какую кодировку должен использовать Web-сервер при работе с файлами — windows-1251, utf-8 или koi8-r;
- **Дополнительные FTP** — на этой странице можно создать дополнительные FTP-входы. Услуга является платной;
- **Web-интерфейс к FTP** — здесь можно загружать файлы на сервер, используя Web-интерфейс;
- **Загрузка архива** — в данном разделе можно закачать архив в форматах zip, rar или tgz. Максимальный размер одного архива — 15 Мбайт. После загрузки архива он автоматически распаковывается на сервере. Обратите внимание: каталог, в который закачивается архив, должен существовать;
- **Управление MySQL** — на этой странице можно создать базу данных MySQL, сменить пароль доступа, а также получить данные для подключения к базе данных из скриптов Perl и PHP. На этой же странице расположена ссылка для доступа к программе PhpMyAdmin;
- **Версия PHP** — здесь можно выбрать версию PHP (4 или 5);

- **Настройки CGI и PHP** — в этом разделе задаются настройки PHP и Perl. Можно задать расширения файлов, включить или отключить выполнение скриптов, а также произвести настройку некоторых директив PHP, например, `magic_quotes_gpc`, `max_execution_time` и др.;
- **Server Side Includes (SSI)** — здесь можно включить или отключить SSI-обработчик, а также указать расширения файлов, которые будут обрабатываться SSI-обработчиком;
- **Crontab** — здесь можно включить или отключить управление автоматическим запуском программ в определенное время;
- **WAP** — в этом разделе можно создать WAP-поддомен. WAP — это протокол, позволяющий получать доступ к разделу сайта, реализованному средствами языка WML, с мобильных устройств;
- **DocumentRoot** — на этой странице можно изменить путь к стартовому каталогу сайта или поддомена. Не рекомендуется изменять этот параметр, если вы не уверены в своих действиях;
- **Архивация сайта** — здесь можно создать архив сайта и баз данных. Архив сайта создается в течение 15 минут и будет находиться в корневом каталоге пользователя. Из соображений безопасности, архив нельзя скачать через Web-браузер. Скачать его можно только по FTP;

❐ **Домены** — здесь производятся все операции над доменами:

- **Список доменных имен** — в данном разделе указаны все домены, зарегистрированные на аккаунте;
- **Продление доменов** — здесь указывается дата окончания регистрации доменных имен. С помощью этого раздела можно продлить домен или установить автоматическое продление доменных имен;
- **Поддомены** — этот раздел позволяет создать поддомен для основного домена. Создать поддомен можно с помощью автоматического создания или вручную. При автоматическом создании поддоменов достаточно создать каталог рядом с каталогом www. Название этого каталога и будет названием поддомена. Например, каталог test автоматически становится доступен из Интернета по адресу http://test.<Имя домена>/. В некоторых случаях нельзя использовать эту опцию, т. к. у автоматических поддоменов переменная окружения `DOCUMENT_ROOT` продолжает указывать на каталог основного сайта. Поэтому необходимо создать поддомен вручную через форму в этом разделе;
- **Данные для регистрации** — здесь можно создать персону для регистрации доменного имени. Мы уже рассматривали создание персоны для регистрации доменных имен в *разд. 6.3*;

- **Заказ домена** — в этом разделе можно зарегистрировать новый домен или перенести существующий;

- **Создание сайта** — здесь можно всего за несколько минут создать собственный полноценный сайт. Движок WordPress, который является основой сервиса, позволяет разместить на сайте как текстовую, так и графическую информацию. За один щелчок мыши можно выбирать индивидуальный дизайн сайта;

- **Статистика** — этот раздел позволяет получить доступ к программам статистики **Webalizer** и **Awstats**:

  - **Статистика Webalizer** — на этой странице можно включить программу Webalizer. После включения будет доступна ссылка для просмотра статистики. Статистика составляется на основе анализа логов сервера и является очень подробной. Для включения программы нажимаем кнопку **Включить**;

  - **Статистика Awstats** — мощная система статистики для сайта. Более 30 видов отчетов. Умеет распознавать фразы, по которым на сайт приходят с поисковиков.

И наконец, практически во всех разделах панели управления присутствует ссылка **Закончить работу**. После завершения работы с панелью управления следует обязательно перейти по этой ссылке.

6.5. Структура каталогов сервера и загрузка контента на сервер

Для доступа к каталогам сервера воспользуемся файловым менеджером WebFTP. Для этого в панели управления переходим на страницу **Сайт | Web-интерфейс к FTP**. Откроется окно, изображенное на рис. 6.6.

Корневая папка сервера содержит следующие каталоги:

- `logs` — содержит журналы (логи), в которые записываются все обращения к сайтам;

- `perlbookspbru` — корневая папка для файлов домена perlbook.spb.ru.

> ***Обратите внимание***
>
> Название папки будет соответствовать вашему домену.

Для перехода в папку perlbookspbru достаточно щелкнуть ее название. В данной папке находится каталог www. Переходим в папку www. Это основная папка для всего содержимого сайта — html-файлов, php-скриптов, изображений и т. п. Именно в этом каталоге должен располагаться файл index.html или

index.php, первый загружаемый файл (центральная страница сайта). Изменить название файла, загружаемого по умолчанию, можно с помощью файла .htaccess или в разделе **Сайт | Индексные файлы** Панели управления.

Внутри каталога perlbookspbru содержится папка cgi-bin. Данная папка предназначена для скриптов, написанных на языке Perl.

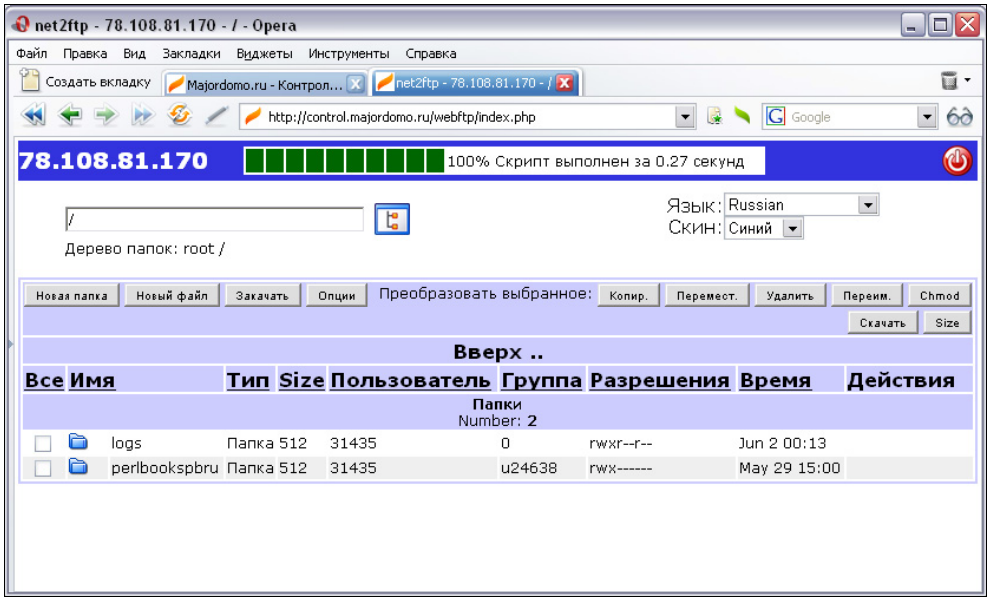


Рис. 6.6. Файловый менеджер WebFTP

> ***Обратите внимание***
>
> Загружать скрипты на Perl необходимо обязательно в текстовом режиме (режим ASCII). После загрузки необходимо изменить права доступа к файлу. Для исполнения скриптов на Perl устанавливаем права 755 (`-rwxr-xr-x`).

Для создания новой папки необходимо нажать кнопку **Новая папка**. В новом окне ввести название папки (или нескольких папок) и щелкнуть на изображении в виде зеленого флажка. Отобразится сообщение о создании папки. Щелкнем на изображении в виде стрелки. Новая папка отобразится в разделе **Папки**.

Для создания нового файла необходимо нажать кнопку **Новый файл**. В поле **Новое имя файла** вводим название файла с расширением, а в текстовое поле вводим любой текст.

Содержимое любого текстового и графического файла можно посмотреть, щелкнув кнопку **Показ.**, для редактирования содержимого файла необходимо щелкнуть на надписи **Редакт.**. Файл можно переименовать или удалить.

Для любого файла можно задать права доступа. Для этого устанавливаем флажок напротив имени файла и нажимаем кнопку **Chmod**.

Для загрузки уже готовых файлов следует нажать кнопку **Закачать**. Отобразится окно, разделенное на две части: **Файлы** и **Архивы**. Загрузить файлы можно в виде архива в формате zip, tar, tgz или gz. При загрузке архива он автоматически распаковывается на сервере. Для загрузки архива в разделе **Архивы** необходимо выбрать его с помощью кнопки **Обзор**. При загрузке файлов следует придерживаться следующих правил:

- центральная страница сайта должна называться index.html или index.php;
- названия всех папок и файлов не должны содержать русских букв (лучше использовать строчные буквы). Помните, что File.html и file.html — это разные файлы! При попытке открыть файл будет выведено сообщение об ошибке 404 (файл не найден);
- внутри каждой папки должен быть расположен файл index.html или index.php. В противном случае при запросе вида **http://vasya.ru/folder1/** будет выведено сообщение об ошибке 403 (нет доступа). Иными словами, в папке folder1 должен быть файл index.html или index.php.

Загрузить файлы можно по протоколу FTP. При помощи FTP-клиентов можно работать с файлами на удаленном компьютере, как будто они расположены на вашем локальном компьютере. FTP-клиент встроен также в некоторые HTML-редакторы, такие как HomeSite, Dreamweaver или FrontPage. Рассмотрим доступ к файлам сервера с помощью популярных FTP-клиентов.

6.5.1. Использование программы CuteFTP 8

CuteFTP, пожалуй, самая удобная и популярная программа для работы с FTP. Для создания нового соединения в меню **File** выбираем пункт **New**. В открывшемся списке выбираем пункт **FTP Site**. Такой же эффект можно получить нажав комбинацию клавиш <Ctrl>+<N>. Откроется окно **Site Properties** (рис. 6.7). На вкладке **General** в поле **Label** вводим любое название, например, `Majordomo`. Это будет название соединения, которое впоследствии отобразится на вкладке **Site Manager**. В поле **Host address** вводим IP-адрес, указанный в письме, которое пришло после регистрации. В поле **Username** вводим свой логин. В поле **Password** можно ввести пароль. Если поле не заполнено, то при подключении нужно будет ввести пароль. В группе переключателей **Login method** следует установить флажок **Normal**.

Переходим на вкладку **Type** (рис. 6.8) и из списка **Data connection type** выбираем пункт **Use PASV**. Нажимаем кнопку **OK**.

Для установления соединения на вкладке **Site Manager** делаем двойной щелчок на созданном названии соединения. В итоге на правой вкладке отобра-

зится содержимое корневого каталога сервера, а слева отобразится содержимое вкладки **Local Drives** (рис. 6.9).

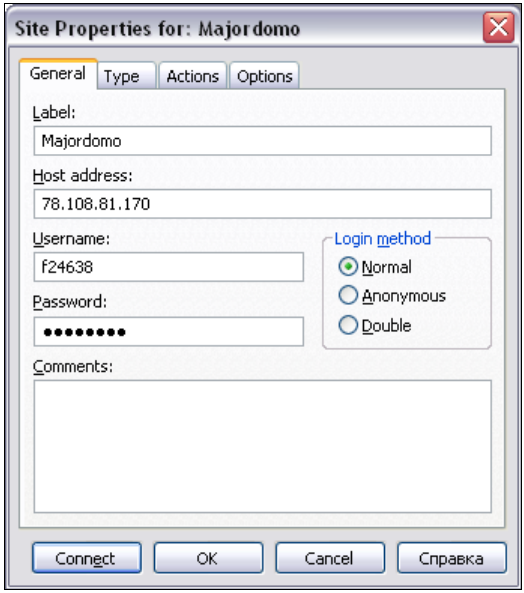


Рис. 6.7. Окно **Site Properties**

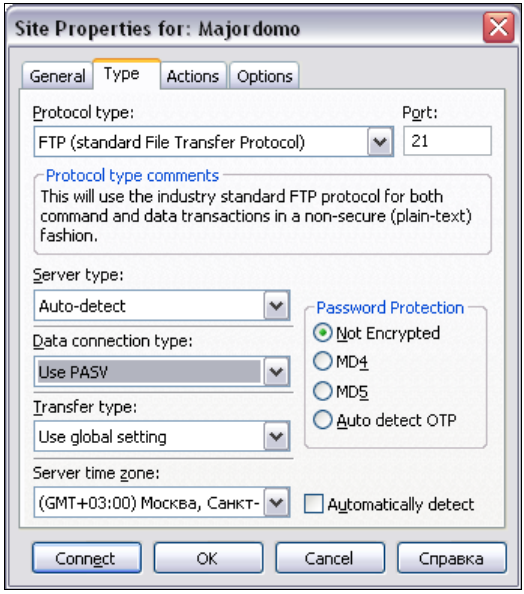


Рис. 6.8. Вкладка **Type** окна **Site Properties**

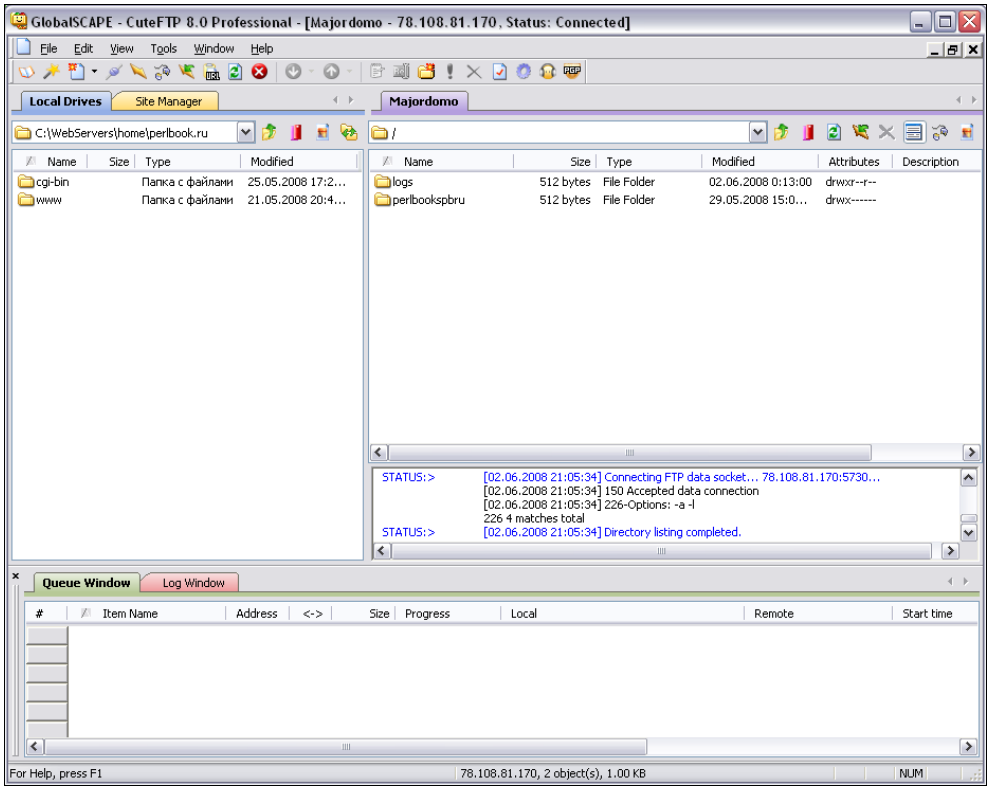


Рис. 6.9. Главное окно программы CuteFTP 8

Справа переходим в папку /<Имя домена>/www. Для этого делаем двойной щелчок вначале на изображении папки <Имя домена>, затем на **www**. Слева выбираем папку с проектом на локальном компьютере.

Для загрузки файла на сервер щелкнем на названии файла правой кнопкой мыши. Из контекстного меню выбираем пункт **Upload**. Можно также воспользоваться кнопкой на панели инструментов с изображением зеленого круга с белой стрелкой, указывающей вверх, или пунктом **Upload** из меню **File**. Самый простой способ — это сделать двойной щелчок мыши на названии файла.

Для загрузки файла с сервера на локальный компьютер щелкнем на названии файла правой кнопкой мыши. Из контекстного меню выбираем пункт **Download**. Можно также воспользоваться кнопкой на панели инструментов с изображением зеленого круга с белой стрелкой вниз или пунктом **Download** из меню **File**. Самый простой способ — это сделать двойной щелчок мыши на названии файла.

Загрузка файлов возможна в двух режимах: ASCII и Binary. Режим ASCII используется для загрузки текстовых файлов, а Binary — для загрузки картинок. Нельзя загружать картинки в режиме ASCII, т. к. это может их повредить. Для выбора режима в меню **File** выбираем пункт **Transfer Type**. В открывшемся списке устанавливаем флажок нужного режима. При установке флажка **Auto** программа будет автоматически определять режим.

Для изменения или удаления файла следует выбрать соответствующий пункт в контекстном меню. Чтобы создать папку, щелкнем правой кнопкой мыши в свободном месте окна и из контекстного меню выберем пункт **New Folder**. Для изменения прав доступа к файлу или каталогу необходимо выбрать из контекстного меню пункт **Properties / CHMOD**.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Загружать скрипты на Perl следует обязательно в режиме ASCII в папку cgi-bin. После загрузки необходимо изменить права доступа к файлу. Для исполнения скриптов на Perl устанавливаем права в 755 (`-rwxr-xr-x`).

6.5.2. Использование программы AceFTP 2

Для создания нового соединения в меню **File** выбираем пункт **Connect**. Откроется окно **Site Profiles**. В меню **File** выбираем пункт **Create**. Из появившегося списка выбираем пункт **New site profile** или нажимаем комбинацию клавиш <Ctrl>+<S>. В первом поле открывшегося окна вводим произвольное название соединения, оно впоследствии отобразится в окне **Site Profiles**. В поле **Server** вводим IP-адрес, указанный в письме. В поле **User ID** вводим свой логин. В поле **Password** можно ввести пароль. Если поле не заполнено или не установлен флажок **Save password**, то при подключении нужно будет ввести пароль. Нажимаем **Finish**. Для установки соединения делаем двойной щелчок мыши на названии.

Загрузка файлов возможна также в режимах ASCII или Binary. Для выбора режима в меню **Tools** выбираем пункт **Default Transfer Mode**. В открывшемся списке устанавливаем флажок нужного режима. При установке флажка **Automatic** программа будет автоматически определять режим.

6.5.3. Использование программы FAR manager

В левой панели FAR manager отображаем содержимое папки с файлами проекта. Затем переключаемся на правую панель с помощью клавиши <Tab>. Нажимаем комбинацию клавиш <Alt>+<F2>. Откроется меню выбора устройств. Выбираем пункт **FTP** и нажимаем <Enter>. Для создания нового соединения нажимаем комбинацию клавиш <Shift>+<F4>. В первой строке от-

крывшегося окна набираем ftp://<Логин>:<Пароль>@<IP-адрес>, например, ftp://f24638:123456@78.108.81.170. Стрелками выбираем пункт **Passive mode** и нажимаем пробел для установки флажка. Стрелками выбираем пункт **Save** и нажимаем <Enter>. Название сайта отобразится на правой панели. Для соединения с сервером выбираем название сайта и нажимаем <Enter>. Содержимое корневой папки сервера отобразится на правой панели. Для копирования файла следует его выделить и нажать клавишу <F5>. Для изменения прав доступа следует выделить файл и нажать комбинацию клавиш <Ctrl>+<A>.

6.6. Настройка Web-сервера Apache с помощью файла .htaccess

На виртуальном хостинге нет доступа к файлу конфигурации httpd.conf. Вместо этого файла необходимо использовать файл .htaccess. Если требуется, чтобы настройки были доступны для всего сайта, то файл .htaccess должен быть расположен в корневой папке. Действие файла .htaccess распространяется на все папки выше по иерархии и не распространяется на папки, расположенные ниже.

> ***Обратите внимание***
>
> Файл .htaccess не имеет названия. Только расширение htaccess.

В файле .htaccess, расположенном в корневой папке, обязательно должны присутствовать директивы обработки следующих ошибок:

- ошибки 404 (файл не найден);
- ошибки 403 (нет доступа);
- если используется защита содержимого папки с помощью сервера Apache, то должна быть обработка ошибки 401 (неавторизован).

```
ErrorDocument 404 <Путь к файлу>
ErrorDocument 403 <Путь к файлу>
ErrorDocument 401 <Путь к файлу>
```

Дизайн указанных файлов должен соответствовать дизайну всего сайта. Все ссылки в этих файлах должны иметь абсолютный URL-адрес. Это касается и адресов картинок. В противном случае получите множество сообщений об ошибке 404.

С помощью директивы `DirectoryIndex` можно задать название файла, который будет выдаваться сервером по умолчанию. Возможно указание нескольких имен файлов через пробел. В этом случае сервер отобразит первый существующий файл из списка. Если ни один файл не найден, то сервер выдаст

ошибку 403. Практически везде названия таких файлов — index.html и index.php:

```
DirectoryIndex default.php default.html
```

Далее следует проверить вывод сообщения об ошибке 403 при запросе каталога без индексного файла. Создайте папку (например, test) и введите в адресной строке Web-браузера **http://<Имя сайта>/test/**. Если в результате получили сообщение об ошибке 403, то все нормально. Если получили листинг каталога, то следует добавить следующую директиву:

```
Options -Indexes
```

В файле .htaccess можно использовать и другие директивы. Например, с помощью директив `Deny` (запретить) и `Allow` (разрешить) можно управлять доступом к объекту. Для ограничения доступа к определенному файлу или группе файлов используется раздел `Files`:

```
<Files *.txt>
   Deny from all
</Files>
```

Символ `*` соответствует любой последовательности символов, а символ `?` — любому одиночному символу. При указании значения `all` в директиве `Deny` доступ к файлу закрыт для всех пользователей. В этом примере мы закрываем доступ к файлам с расширением TXT. Теперь при попытке получить запрещенный файл сервер выдаст сообщение об ошибке 403 (нет доступа).

6.7. Файл favicon.ico

Когда посетители добавляют сайт в Избранное, Web-браузеры запрашивают с сервера файл favicon.ico. Этот файл должен содержать логотип сайта в виде иконки 16×16 или 32×32. В одном файле может находиться несколько форматов иконки сразу. Если файл не найден, то в журнал регистрации ошибок записывается сообщение об ошибке 404 (файл не найден), а в строке в Избранном отобразится стандартная иконка Web-браузера. Если иконка найдена, то она отобразится в Избранном и в адресной строке Web-браузера перед URL-адресом. Некоторые поисковые порталы (например, Яндекс) отображают иконку в результатах поиска.

Файл favicon.ico должен находиться в корневой папке сервера, а в раздел `HEAD` HTML-документа можно добавить следующие строки:

```
<LINK rel="icon" href="http://<Имя сайта>/favicon.ico"
type="image/x-icon">
<LINK rel="shortcut icon" href="http://<Имя сайта>/favicon.ico"
type="image/x-icon">
```

Файлы создаются в специализированных редакторах. Дополнительную информацию о файле favicon.ico и методах его создания можно получить на сайтах **http://favicon.ru/** и **http://favicon.com/**.

6.8. Файл robots.txt

Еще один файл, отсутствие которого приводит к ошибке 404 (файл не найден), называется robots.txt. Перед индексацией сайта все роботы обязаны ознакомиться с содержимым этого файла. Даже если все страницы сайта разрешены для индексации, все равно следует создать пустой файл robots.txt и разместить его в корневой папке сайта. Отсутствие файла означает, что индексация разрешена.

В некоторых случаях следует запретить индексацию отдельных страниц сайта или всего сайта, например, если он предназначен только для членов семьи. Следует также учитывать, что поисковые роботы создают большой трафик, т. к. рекурсивно запрашивают все страницы сайта без передышки. Это обстоятельство может значительно нагрузить сервер.

Для снижения нагрузки на сервер необходимо в корневой папке сайта создать файл robots.txt. Содержимое файла выглядит следующим образом:

```
User-agent: *
Disallow: /file.html
Disallow: /user/
```

В директиве `User-agent` можно задать конкретное название робота или указать символ `*`, который означает, что данная информация относится ко всем роботам. Узнать название робота можно по полю `user-agent` в логах сервера.

Каждая строка `Disallow` определяет файл или каталог, который запрещен для индексации указанным роботом.

Чтобы запретить индексацию всего сайта, необходимо разместить файл со следующими директивами:

```
User-agent: *
Disallow: /
```

6.9. Защита содержимого папки

Для защиты содержимого папки с помощью сервера Apache необходимо разместить в этой папке файл .htaccess со следующими директивами:

```
AuthType Basic
AuthName "Enter password"
```

```
AuthUserFile /home/<Логин>/include/.htpasswd
<Limit GET POST>
   require valid-user
</Limit>
```

В директиве `AuthName` можно указать любой текст. Он будет отображен в диалоговом окне в качестве подсказки.

> ***ОБРАТИТЕ ВНИМАНИЕ***
> Ваш логин можно найти на странице **Сайт | .htaccess и .htpasswd**.

Теперь необходимо создать файл .htpasswd и разместить его в папке include. Для создания этого файла заходим в панель управления на страницу **Сайт | .htaccess и .htpasswd**. Заполняем поле **Имя пользователя**, а также поле **Пароль пользователя**. Остальные поля можно не заполнять. Нажимаем кнопку **Сгенерировать текст файлов**. На самом деле будут сгенерированы оба файла. Нас интересует только созданный файл .htpasswd с логином и паролем пользователя. Копируем содержимое текстового поля **Текст файла .htpasswd**. Открываем Блокнот и вставляем содержимое буфера обмена. Затем сохраняем файл под названием .htpasswd и загружаем его на сервер в папку include.

> ***ОБРАТИТЕ ВНИМАНИЕ***
> Если вы не создавали папку include ранее, то ее необходимо создать.

Теперь при попытке войти в защищенную папку будет выведено окно для ввода логина и пароля (рис. 6.10).

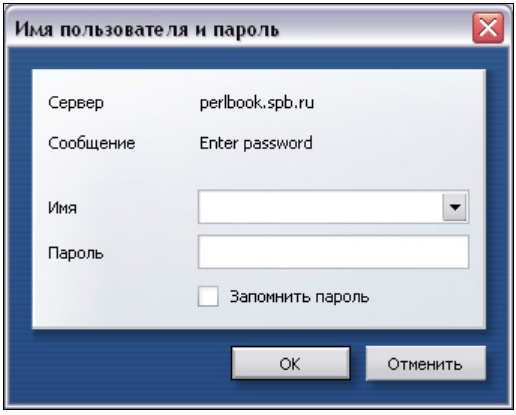


Рис. 6.10. Окно для ввода логина и пароля

Если необходимо создать пароль для другого пользователя, то опять генерируем логин и пароль, а затем добавляем полученный код в конец файла

.htpasswd. Для запрета входа определенному пользователю достаточно удалить строку об этом пользователе из файла .htpasswd.

6.10. Создание базы данных MySQL

На виртуальном хостинге нет возможности создать базу данных через phpMyAdmin или через SQL-запрос. Для создания базы данных необходимо зайти в панель управления хостинга и выбрать пункт **Сайт | Управление MySQL**. В открывшемся окне отобразится информация о созданных базах данных, а также данные для подключения из скриптов Perl и PHP.

Для создания базы данных заполняем форму **Создать базу данных** и нажимаем кнопку **Создать**.

Для создания нового пароля необходимо заполнить форму **Смена пароля на MySQL**. Вводим новый пароль и повторяем его. Затем нажимаем кнопку **Сохранить изменения**. Для примера создадим пароль "123456".

На этой же странице есть ссылка для доступа к программе phpMyAdmin.

6.11. Управление базой данных при помощи phpMyAdmin

Для доступа к программе phpMyAdmin заходим в панель управления на страницу **Сайт | Управление MySQL**. Затем переходим по ссылке в раздел **Интерфейс для доступа к базам данных PhpMyAdmin**. Откроется страница для ввода логина и пароля. Заполняем поля и нажимаем кнопку **OK**. Откроется главная страница phpMyAdmin (рис. 6.11).

Окно программы phpMyAdmin разделено на две части. Слева расположен список с доступными базами данных и несколько иконок сверху, из которых нас интересует только иконка с надписью **Exit** (рис. 6.12) выхода из программы.

Для перехода к базе данных необходимо выбрать ее из списка. Под списком отобразится перечень всех таблиц, а справа — перечень таблиц с панелью доступных действий и статистикой.

Для создания таблицы можно выполнить SQL-запрос на вкладке **SQL** или воспользоваться возможностями программы. Для примера создадим таблицу `city`. Из списка выбираем нашу базу данных. В поле **Создать новую таблицу в БД** вводим название таблицы, а в поле **Количество полей** вводим количество полей в создаваемой таблице — 2. Нажимаем кнопку **OK**. Отобразится два поля для ввода.

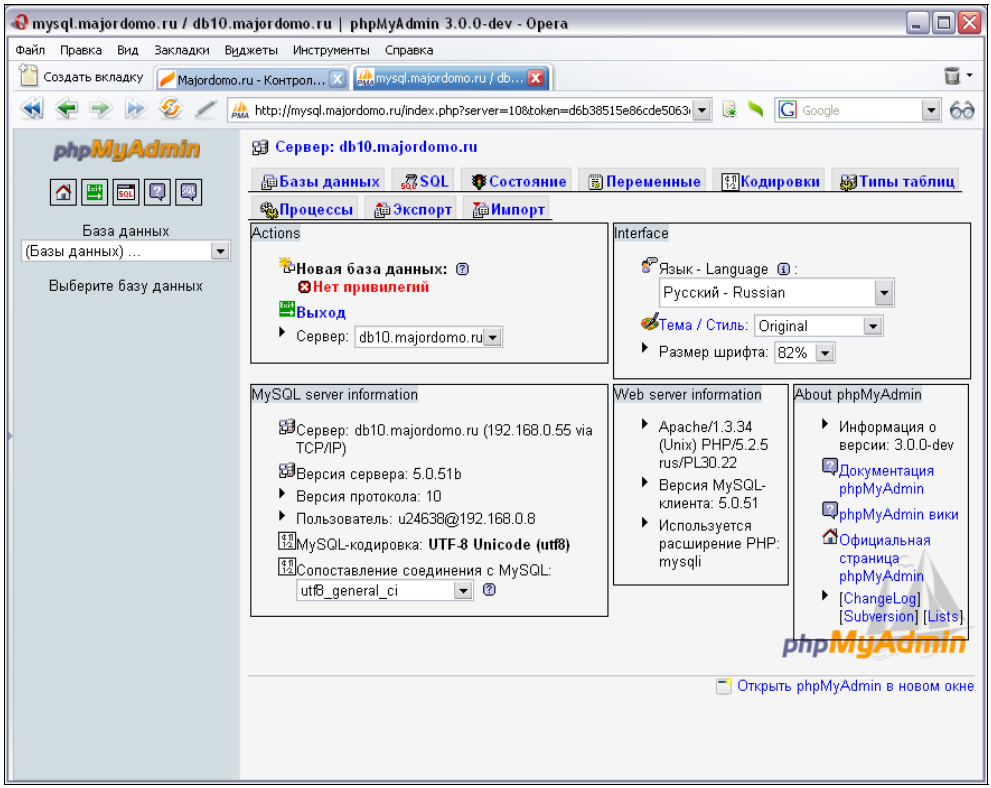


Рис. 6.11. Главное окно программы phpMyAdmin

Рис. 6.12. Элементы левой части окна программы phpMyAdmin

В первое поле вводим `id_city`. Из списка **Тип** выбираем пункт **INT**. Устанавливаем флажок напротив пункта **Первичный ключ**, а из списка **Дополнительно** выбираем пункт **auto_increment**. Во второе поле вводим `name_city`. Из списка **Тип** выбираем пункт **CHAR**, а в поле **Длины** вводим число `250`. Из списка **Сравнение** выбираем пункт **cp1251_general_ci**. Под списком полей доступны дополнительные опции. Из списка **Тип таблиц** выбираем пункт **MyISAM**, а из списка **Сравнение** выбираем пункт **cp1251_general_ci**. Нажимаем кнопку **Сохранить**.

Сверху над структурой базы данных расположено несколько вкладок. Из них нас интересует вкладка **SQL** для выполнения SQL-запросов, а также вкладка **Экспорт** для создания резервной копии базы данных. Сделать копию можно в нескольких форматах. Из всех форматов следует выбрать SQL, установить флажок **Сохранить как файл** и нажать кнопку **OK**. Отобразится окно с запросом **Что делать с файлом?**. Нажимаем кнопку **Сохранить** и выбираем место для сохранения дампа базы данных. В итоге будет создан файл с расширением sql.

Если база данных имеет большой объем, то следует делать дамп отдельно для каждой таблицы. Для этого на левой панели щелкнем на названии таблицы. Справа отобразится структура полей таблицы, изменится структура вкладок.

Выбираем вкладку **Экспорт**. Устанавливаем флажок **SQL**. Устанавливаем флажок **Сохранить как файл** и нажимаем кнопку **OK**. Основное отличие от предыдущего метода заключается в возможности сохранить не все записи за один раз, а только определенное количество. Для этого существуют два поля **Дамп из строк, начиная с записи #**.

Для восстановления таблицы из сохраненного файла следует указать этот файл на вкладке **Импорт**. Предварительно нужно удалить таблицу или при создании дампа следует установить флажок **Добавить DROP TABLE**. В этом случае в файл будет добавлена строка:

```
DROP TABLE IF EXISTS <Название таблицы>;
```

Она позволяет автоматически удалить таблицу.

Все SQL-запросы на конкретную таблицу следует осуществлять, выбрав эту таблицу, а затем перейдя на вкладку **SQL**. Справа отобразится список всех полей таблицы. Двойной щелчок позволяет вставить название поля в текстовую область.

Перейдя на вкладку **Обзор** можно просмотреть содержимое таблицы и при необходимости удалить или отредактировать запись.

Для добавления нового поля в уже существующую таблицу необходимо на вкладке **Структура** в поле **Добавить поле(я)** ввести количество добавляемых полей. Поле можно вставить в конец или начало таблицы, а также после определенного поля, выбрав его из списка.

> ***Обратите внимание***
>
> Добавляемое поле должно допускать значение `Null` или иметь значение по умолчанию, т. к. в таблице уже есть данные.

6.12. Доступ к базе данных с помощью Perl

В качестве примера рассмотрим доступ к базе данных из скриптов, написанных на языке Perl. В предыдущем разделе мы создали таблицу `city` с двумя полями — `id_city` и `name_city`. Используем эту таблицу в нашем примере.

Для доступа к базе данных необходимо на странице **Сайт | Управление MySQL** получить данные для подключения из скриптов. В моем случае эти данные следующие:

- хост — 78.108.81.250;
- пользователь — u24638;
- название базы данных — b24638;
- пароль — 123456;
- название таблицы — city.

Прежде чем создавать программы на Perl, необходимо узнать местоположение интерпретатора. Обычно эта информация находится в разделе "Инструкции" или "FAQ". В случае хостинга Majordomo информация расположена в разделе **Помощь | Ответы на часто задаваемые вопросы (FAQ) | Где у Вас находится интерпретатор Perl, sendmail?**:

```
/usr/bin/perl
/usr/sbin/sendmail
```

Первая строка это путь к интерпретатору, а вторая к программе Sendmail. Путь к программе Sendmail понадобится при отправке писем с помощью Perl.

Добавим несколько городов в таблицу `city` и выведем все города в обратном алфавитном порядке. Открываем Блокнот и набираем следующий код:

```
#!/usr/bin/perl -w
use DBI;
print "Content-type: text/html\n\n";

my $host = "78.108.81.250";
my $db_name = "b24638";
my $user = "u24638";
my $password = "123456";

my $db = DBI->connect("DBI:mysql:$db_name:$host", $user, $password,
         { RaiseError => 1 });
$db->do("insert into city values(NULL, 'Санкт-Петербург')");
$db->do("insert into city values(NULL, 'Москва')");
$db->do("insert into city values(NULL, 'Новгород')");
```

```
$db->do("insert into city values(NULL, 'Самара')");
$db->do("insert into city values(NULL, 'Псков')");

my $query = "select name_city from city order by name_city desc";
my $result = $db->prepare($query);
$result->execute();
while (my @pole = $result->fetchrow_array()) {
   print "$pole[0]<BR>";
}
$result->finish();
$db->disconnect();
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Не забудьте заменить данные для подключения своими.

Сохраняем под названием testdb.pl и загружаем файл на сервер через FTP в папку cgi-bin. В моем случае это /perlbookspbru/cgi-bin.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Загружать файл необходимо обязательно в режиме ASCII. После загрузки следует изменить права доступа к файлу. Для исполнения скриптов на Perl устанавливаем права в 755 (`-rwxr-xr-x`). Изменить права доступа позволяет практически любой FTP-клиент.

Открываем Web-браузер и в адресной строке набираем:

```
http://<Название домена>/cgi-bin/testdb.pl
```

В результате получим:

```
Санкт-Петербург
Самара
Псков
Новгород
Москва
```

Заключение

Вот и закончилось наше путешествие в мир Perl. Материал этой книги лишь вводит в язык Perl и не охватывает все его возможности. Мир Perl это не только ядро, но и множество самых разнообразных модулей, созданных сообществом программистов и доступных для свободного скачивания. Если описывать эти модули, то издать подобную книгу не представляется возможным. В этом заключении мы рассмотрим, где можно найти дополнительную информацию и продолжить изучение языка Perl.

Первым и самым важным источником информации о ядре Perl и всех тонкостях его работы является книга "Программирование на Perl" (3-е изд., 1152 стр., издательство "Символ-Плюс"). Одним из авторов этой книги является создатель языка Perl Ларри Уолл. Это не просто описание Perl, а уникальное введение в культуру языка. По праву она считается "библией" Perl.

Еще одним источником информации является электронная документация. Чтобы отобразить полный перечень доступной документации, необходимо запустить программу cmd.exe и в командной строке ввести команду:

```
perldoc perl
```

Программа perldoc допускает использование нескольких флагов. Например, с помощью флага `-f` можно произвести поиск в описаниях функций, а флаг `-q` производит поиск в списке часто задаваемых вопросов:

```
perldoc -f exit
perldoc -q exit
```

Чтобы вывести описание всех флагов, набираем команду:

```
perldoc -h
```

Внутри модулей также размещается документация. Все установленные в системе модули можно найти в каталогах C:\WebServers\usr\local\perl\lib и

C:\WebServers\usr\local\perl\site\lib. Чтобы отобразить документацию из модуля, в командной строке набираем:

```
perldoc <Название модуля>
```

Например, выведем документацию из модуля `URI`:

```
perldoc URI
```

Чтобы получить документацию формата POD в формате HTML, можно воспользоваться следующей командой:

```
pod2html C:/WebServers/usr/local/perl/lib/URI.pm > C:/WebServers/tmp.html
```

Теперь можно просматривать документацию к модулю, открыв файл tmp.html в любом Web-браузере.

Важным подспорьем будет также CPAN (Comprehensive Perl Archive Network). Зеркальные серверы CPAN разбросаны по всему Интернету. Попасть в архив CPAN можно, например, набрав **http://cpan.perl.org/** или **http://cpan.org/**. Произвести поиск нужного модуля можно на сайте **http://search.cpan.org/**.

Регулярно посещайте следующие сайты:

- **http://perl.org/**;
- **http://www.perl.com/**;
- **http://www.activestate.com/**;
- **http://dev.perl.org/**;
- **http://perldoc.perl.org/**.

И наконец, не следует забывать, что часто достаточно в строке запроса поискового портала (например, **http://yandex.ru/**, **http://www.google.com/**, **http://www.rambler.ru/**) ввести свой вопрос или обратиться к специальным форумам по Perl (например, **http://forum.vingrad.ru/Perl-forum.html**). Задать вопрос можно также на форуме моего сайта **http://wwwadmin.ru/forum/**.

Предметный указатель

Л

М

О

П

Р

С

Т

Ф

Х

Ц

Я